Microsoft Visual C#

Esta apostila apresenta diferentes ferramentas e técnicas do Microsoft Visual C#. O estudante aprenderá a declarar variáveis, escrever instruções, criar classes e métodos; construir interfaces gráficas, gerenciar erros e tratar exceções, acessar banco de dados; além de apresentar orientação prática e exemplos para criar aplicativos com a linguagem C#.

Prof. Msc. Regilan Meira
IFBA – Campus Ilhéus

Sobre este material

Esta apostila tem como objetivo auxiliar os estudantes de escolas técnicas, na aprendizagem de disciplinas relacionadas a programação de computadores, cuja a linguagem e ambiente de programação é o Visual C#, presente nos cursos técnicos da área de Informática e Tecnologia da Informação. Esta apostila não substitui os livros, sua finalidade é criar um roteiro resumido do ensino-aprendizagem realizado em sala de aula.

Este material foi construído a partir de slides de aulas, apostilas, tutoriais, dicas e demais fontes de dados, obtidos a partir de pesquisas em sites de buscas na Internet. Além disto, este material tem como referência a seguinte bibliografia:

GREENE, Jennifer; STELLMAN, Andrew. Use a cabeça C#. 2ª Edição, Editora Altabooks: 2010

MANZANO, José Augusto N G. Estudo dirigido de Microsoft Visual C# 2012. 1ª Edição, Editora Érica: 2012

SHARP, John. Microsoft Visual C# 2010 - Passo a Passo. 1ª Edição, Editora Bookman: 2011

Sumário

1. Conceitos gerais 4

1.1. Primeira Aplicação 4

2. Declaração de variáveis e operadores 9

2.1. Variáveis 9

2.2. Constantes 12

2.3. Funções de conversão e operadores 13

2.4. Exercícios 14

3. Estruturas de decisão e de repetição 17

3.1. Estruturas de decisão 17

3.2. Operadores Relacionais e Operadores Lógicos 20

3.3. Estruturas de repetição 23

3.4. Exercícios 27

4. Arrays e Funções 33

4.1. Arrays 33

4.2. Funções 35

5. Enumeradores, Modificadores de Acesso e Structs 41

5.1. Enumeradores 41

5.2. Modificadores de Acesso 42

5.3. Structs 43

5.4. Exercícios 45

6. Programação orientada a objetos: classes e métodos 47

6.1. Classes e objetos: conceitos 48

6.2. Classes e Objetos: Implementação no C# 49

6.3. Herança 57

6.4. Polimorfismo 64

6.5. Exercícios 66

7. Interface gráfica com usuário (GUI) 69

7.1. Controle Label, TextBox, Button 73

7.2. ComboBox, listBox, checkbox, radiobutton, maskedtextbox e controles para datas 76

7.3. GroupBox, panel, tabcontrol, listview, treeview, e outros 84

7.4. Menustrip, toolstrip e statusbartrip 92

8. Acesso a dados 98

9. Extras: Exercícios propostos para construção de interfaces gráficas 106

1. Conceitos gerais

C# (lê-se CSharp) é uma linguagem de programação orientada a objetos desenvolvida pela Microsoft como parte de sua plataforma de desenvolvimento de sistemas. A sua sintaxe orientada a objetos foi baseada no C++ mas inclui influencias de outras linguagens de programação, como o Delphi e o Java.

Para desenvolver aplicativos em C# necessitamos de um ambiente integrado para desenvolvimento de software (IDE ou Integrated Development Environment) que suporte a linguagem C# e a .Net Framework (pacote de desenvolvimento de aplicativos para plataforma .Net). A IDE oficial de desenvolvimento em C# é o Visual Studio 2012, que é uma ferramenta da Microsoft para o desenvolvimento de aplicativos. Para adquirir uma das diversas versões do Visual Studio 2012 é necessário adquirir uma licença, porém a Microsoft distribui uma série de ferramentas gratuitas de desenvolvimento conhecida como *Express*, no qual podemos fazer o download diretamente do site da Microsoft, registrar o software e iniciar o desenvolvimento de nossos aplicativos. Neste estudo iremos adotar a ferramenta Visual Studio Express 2012 para Windows Desktop, que pode ser acessada através do link: http://www.microsoft.com/visualstudio/ptb/downloads#d-express-windows-desktop

Além do download da ferramenta de desenvolvimento para Windows Desktop, podemos fazer o download de outras ferramentas gratuitas como Visual Studio Express 2012 para Web, Windows 8 e Windows Phone, além do Microsoft SQL Server Express Edition.

A instalação dos aplicativos é bastante prática e segue o método tradicional de instalação de aplicativos Windows: ***Avançar > Avançar > Concluir***. Depois de instalado, será criado um atalho para o programa no **Menu Iniciar.**

1.1. Primeira Aplicação

Após instalação e registro do Visual Studio Express 2012 para Windows Desktop, iremos para nossa primeira aplicação. É comum quando aprendemos uma nova linguagem de programação começar nosso primeiro programa com a saudação

"ALÔ, MUNDO". Nesta apostila não iremos fazer diferente, e nosso primeiro programa será uma aplicação console apresentando a famosa fase.

Inicie o Visual Studio Express 2012 para Windows Desktop e crie um novo projeto – *File* > *New Project*. Na janela a seguir escolha a linguagem de programação, neste caso Visual C# e selecione o template ***"Console Application"*** e dê o nome ao projeto de "AloMundo".

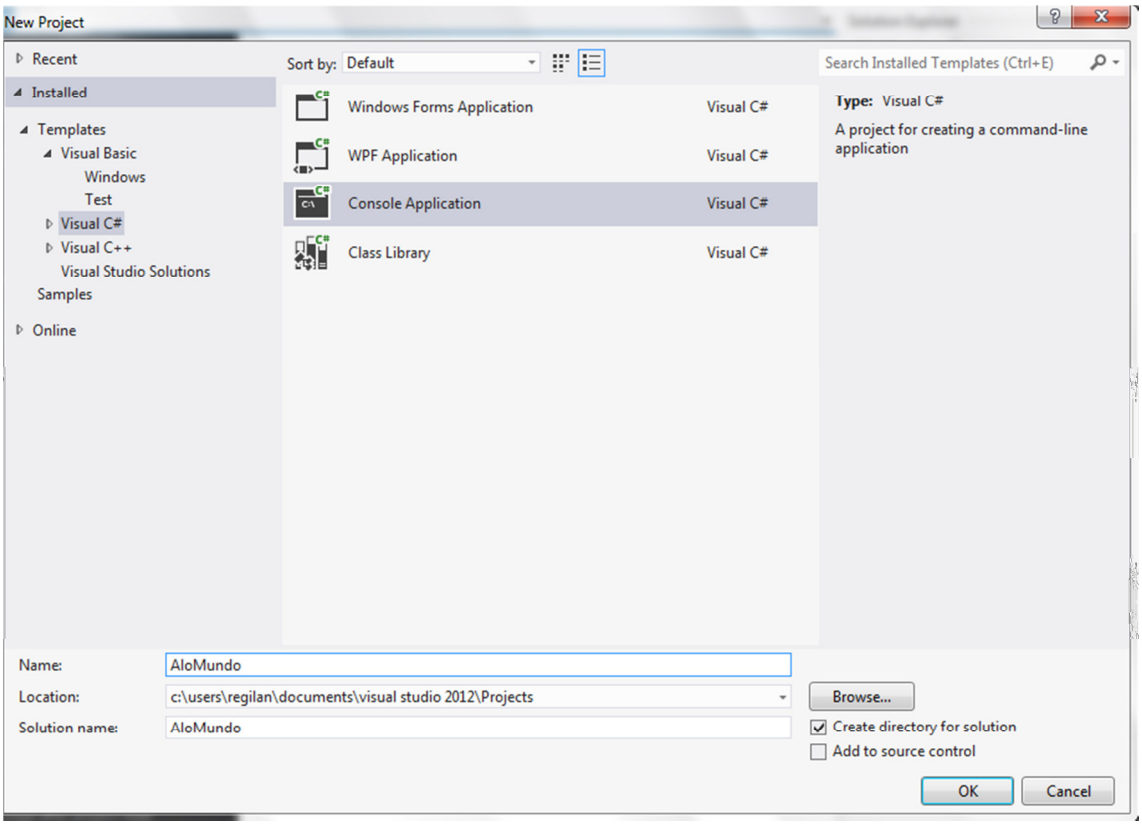


Figura 1 - Novo projeto: tipo Console Application

Assim que o projeto for criado, será mostrado uma janela, contendo um código fonte padrão em C#. Um programa em C# deve conter um método ***Main***, que indica o início da aplicação. Nele você pode cria objetos e executar outros métodos. O método Main pode não retornar valores: Com os dois tipos acima, o método Main pode também ter parâmetros:

Todas as instruções em C# assim como na linguagem C e C++ devem estar entre chaves e sempre ser finalizadas com um ponto-e-vírgula (;) veja no exemplo abaixo:

```
      {
      // As instruções devem estar sempre entre chaves e
finalizadas com ";"
      }
```

A entrada e saída de dados em um programa C# é realizado pela biblioteca de classes do .NET Framework.

Para exibir um dado no aplicativo(saída de dados) console usamos a instrução:

```
      System.Console.WriteLine("TEXTO A SER EXIBIDO")
```

A instrução System.Console.WriteLine("Alô Mundo!"); utiliza o método WriteLine da classe Console. Veja como ficaria a aplicação "AloMundo"

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

namespace AloMundo.cs
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Alô mundo?");
            System.Threading.Thread.Sleep(5000);
        }
    }
}
```

Para a entrada de dados utilizamos o método System.Console.ReadLine(). Vamos apresentar outro exemplo para ajudar a fixar a entrada e saída de dados. Neste próximo exemplo que vamos criar, a aplicação irá perguntar o nome e o ano de nascimento de uma pessoa. Em seguida o programa perguntará a sua idade. Veja o código fonte abaixo:

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

namespace AloMundo.cs
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Qual o seu nome?");
            string leitura = Console.ReadLine();
            string nome = leitura;
            Console.WriteLine("Em que ano você nasceu?", nome);
            leitura = Console.ReadLine();
            int ano = Convert.ToInt32(leitura);
            int idade = DateTime.Today.Year - ano;

            Console.WriteLine("{0}, você tem {1} anos.", nome, idade);
            System.Threading.Thread.Sleep(5000);
        }
    }
}
```

A leitura é efetuada através do método **ReadLine()** que permite recolher toda a cadeia de caracteres que o usuário introduziu até pressionar a tecla "Enter".

A saída é efetuada através do método **WriteLine()** que permite receber qualquer número de parâmetros, sendo que o primeiro é sempre a string de saída, instanciá-los com os argumentos por posição, Ex. {0} corresponderá ao primeiro argumento dado para instanciação, e por fim mostrar a string instanciada.

O método **Sleep()** permite fazer uma pausa, em milisegundos, antes do processo continuar. Neste caso serve para fazer uma espera de 5 segundos antes da aplicação terminar.

Neste exemplo utilizamos também a declaração de algumas variáveis e um método da classe DateTime (DateTime.Today.Year) para saber qual o ano atual (Today.Year) e subtrair pelo ano de nascimento do usuário (ano) para obter a idade atual do individuo.

Observe também que no início do código temos o carregamento do namespace System (**using System;**) que contém a classe Console que estamos utilizando no nosso programa. Em C# usamos a diretiva using seguida do nome do pacote que queremos utilizar. NameSpace no .NET tem a função de organizar as classes, para que não se misturem com outras classes de mesmo nome. São equivalentes aos pacotes do Java.

No nosso aplicativo temos ainda a declaração da nossa classe (***class Program***), neste caso Program. Se você ainda não está familiarizado com o conceito de classes não se preocupe, pois falaremos sobre Programação Orientada a Objetos (POO) nesta apostila de C#. Mas caso você tenha curiosidade sobre programação orientada a objeto recomendo fazer uma pesquisa e iniciar o estudo, pois será muito importante quando chegarmos em POO você tenha um conhecimento inicial sobre essa técnica de programação.

Nesta primeira aplicação abordamos os conceitos primitivos (entrada e saída de dados), presentes em qualquer linguagem de programação. Nas seções a seguir abordaremos a sintaxe da linguagem de programação C#, desde os conceitos fundamentais, como declaração de variáveis e estruturas de decisão e de controle, até a programação Orientada a Objetos.

Este curso está organizado em duas partes: a primeira que envolve a sintaxe da linguagem C# será abordado utilizando exemplos em CONSOLE APPLICATION. A segunda parte da apostila apresentará o desenvolvimento de aplicações visuais para Windows Desktop e conexão com banco de dados.

2. Declaração de variáveis e operadores

2.1. Variáveis

Neste capítulo iremos conhecer os principais tipos de variáveis e operadores. Para essa primeira parte do curso que se refere à sintaxe da linguagem C# iremos trabalhar com programas do tipo Console Application, pois é melhor forma de fixar uma nova linguagem. Ao longo do curso teremos a oportunidade de trabalhar com programas para Windows Desktop.

Durante a execução de um programa, os dados são armazenados temporariamente na memória. **Uma variável** é um nome que se dá a um local na memória que armazena certo tipo de dado, assim cada variável esta associada com um tipo de dado e um valor. A linguagem C#, assim como toda linguagem de programação faz o uso de variáveis e exige que todas as variáveis sejam declaradas.

Toda variável deve ter um nome e tipo que define qual o tipo de dado que deverá ser armazenada, dessa forma toda variável possui regras que determinam seu uso. Podemos dizer que existem cinco categorias básicas de variáveis: inteiros, números de ponto flutuante, booleanas, datas e strings (variáveis do tipo texto).

Nas linguagens de programação orientada a objetos existem dois tipos diferentes de variáveis: os tipos intrínsecos à linguagem, também conhecido como tipos primitivos, e aqueles que são criados pelos programadores, conhecidos como classes.

No C# a declaração de variáveis é obrigatória, assim como a declaração de seus tipos. A declaração de variáveis segue o seguinte formato:

```
[TIPO DA VARIÁVEL] [NOME DA VARIAVEL] = [VALOR];
OU
[TIPO] [NOME DA VARIAVEL];
```

No código a seguir temos um exemplo de declaração e atribuição de valor a uma variável.

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

```
{

string nome = "REGILAN MEIRA SILVA"; //DECLARAÇÃO E ATRIBUIÇÃO DE VARIÁVEL
Console.WriteLine("OLÁ " + nome);
System.Threading.Thread.Sleep(5000);
}
```

A tabela abaixo resume os principais tipos de variáveis em CTS (*Common Type Specification*), que são os tipos comuns presentes no framework .Net, dessa forma podemos utilizar os tipos comuns em qualquer linguagem que suporta a .Net Framework)

Tipo C#	Tipo .NET(CTS)	Faixa de dados	Descrição
bool	System.Boolean	*true* ou *false*	Booleano
byte	System.Byte	-127 a 128	Inteiro de 8-bit com sinal
char	System.Char	U+0000 a U+ffff	Caracter Unicode de 16-bit
decimal	System.Decimal	$1,0 \times 10^{-28}$ a $7,9 \times 1028$	Inteiro de 96-bit com sinal com 28-29 dígitos significativos
double	System.Double	$\pm 5,0 \times 10^{-324}$ a $\pm 1,7 \times 10308$	Flutuante IEEE 64-bit com 15-16 dígitos significativos
float	System.Single	$\pm 1,5 \times 10^{-45}$ a $\pm 3,4 \times 1038$	Flutuante IEEE 32-bit com 7 dígitos significativos
int	System.Int32	-2.147.483.648 a 2.147.483.647	Inteiro de 32-bit com sinal
long	System.Int64	−9,223,372,036,854,775,808 a 9,223,372,036,854,775,807	Inteiro de 64-bit com sinal
Object	System.Object		Classe base
Sbyte	System.Sbyte	0 a 255	Inteiro de 8-bit sem sinal
Short	System.Int16	-32,768 a 32,767	Inteiro de 16-bit com sinal
String	System.String		String de caracteres Unicode
Uint	System.UInt32	0 a 4,294,967,295	Inteiro de 32-bit sem sinal
Ulong	System.UInt64	0 a 18,446,744,073,709,551,615	Inteiro de 64-bit sem sinal
Ushort	System.UInt16	0 a 65,535	Inteiro de 16-bit sem sinal

Figura 2 - Tipos comuns de dados

No código a seguir temos mais alguns exemplos de declaração de variáveis. Na linguagem C#, quando uma variável é declarada e não utilizada, ou mesmo quando a variável é utilizada e não tem seu valor inicial definido, o ambiente de

programação Visual Studio, identifica esta situação como um Warning (Perigo), grifando e indicando na janela de erros (Error List), na parte inferior da ferramenta.

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

namespace Aula_02
{
    class Program
    {
        static void Main(string[] args)
        {
            int idade = 30; //DECLARAÇÃO DE VARIÁVEL
            string sexo = "M"; //DECLARAÇÃO DE VARIÁVEL
            string filhos = "N"; //DECLARAÇÃO DE VARIÁVEL
            double salario = 1897.56; //DECLARAÇÃO DE VARIÁVE
            string nome = "João Vinicius"; //DECLARAÇÃO DE VARIÁVE

        }
    }
}
```

Error List

0 Errors | 5 Warnings | 0 Messages | Search Error List

	Description	File	Line	Colu...	Project
1	The variable 'idade' is assigned but its value is never used	Program.cs	12	17	AloMundo.cs
2	The variable 'sexo' is assigned but its value is never used	Program.cs	13	20	AloMundo.cs
3	The variable 'filhos' is assigned but its value is never used	Program.cs	14	20	AloMundo.cs
4	The variable 'salario' is assigned but its value is never used	Program.cs	15	20	AloMundo.cs
5	The variable 'nome' is assigned but its value is never used	Program.cs	16	20	AloMundo.cs

Figura 3 - Janela Lista de Erros

No próximo exemplo faremos um programa para ler dados do usuário e armazenar os valores em variáveis. Em seguida iremos exibir na tela os dados informados pelo usuário. Veja o resultado:

```
static void Main(string[] args)
        {
            int idade; //DECLARAÇÃO DE VARIÁVEL
            char sexo; //DECLARAÇÃO DE VARIÁVEL
            char filhos; //DECLARAÇÃO DE VARIÁVEL
            double salario; //DECLARAÇÃO DE VARIÁVE
            string nome; //DECLARAÇÃO DE VARIÁVE
            Console.WriteLine("SISTEMA DE ENTRADA DE DADOS\n"); // \n pula uma
linha
```

```
            Console.WriteLine("NOME:");
            nome = Console.ReadLine();
            Console.WriteLine("SEXO (M/F):");
            sexo = Convert.ToChar(Console.ReadLine());
            Console.WriteLine("FILHOS?(S/N)"); //\n pula uma linha
            filhos = Convert.ToChar(Console.ReadLine());
            Console.WriteLine("SALARIO:"); //\n pula uma linha
            salario = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("IDADE:"); //\n pula uma linha
            idade = Convert.ToInt32(Console.ReadLine());
            // Usamos o operador "+" para concatenar os dados e exibir na tela
            Console.WriteLine(nome + ", SEXO: " + sexo + ", IDADE: " + idade + ",
SALARIO: " + salario + ", FILHOS: " + filhos);
            System.Threading.Thread.Sleep(5000);


        }
```

O C# é case-sensitive, então a variável "nome" é diferente de "Nome" e "System.Console.WriteLine()" é diferente de "system.console.writeline()".

No código acima usamos o operador "+" para concatenar strings e mostrar os dados na tela.

```
Console.WriteLine(nome + ", SEXO: " + sexo + ", IDADE: " + idade + ",
SALARIO: " + salario + ", FILHOS: " + filhos);
```

No código apresentado acima, estamos usando funções de conversão pois o dado informado pelo usuário é lido com "STRING", dessa forma faz-se necessário converter para o tipo da variável que é usada para armazenar o valor.

```
idade = Convert.ToInt32(Console.ReadLine());
```

2.2. Constantes

Constante como próprio nome sugere são variáveis que uma vez definida seu valor ela não poderá ser mudado pelo programa. U**ma constante deve sempre ser inicializada em sua declaração, caso contrário será gerado um erro.** Em C# declaramos constantes com a palavra reservada ***"const"***, veja exemplo a seguir:

```
const double pi = 3.1416; //DECLARAÇÃO DA CONSTANTE PI
            Console.WriteLine("PROGRAMA PARA CALCULAR A AREA DE UM
CIRCULO\n");
            double area,raio; //DECLARAÇÃO DE DUAS VARIAVEIS DO MESMO
TIPO PODEM SER FEITA NA MESMA LINHA
            Console.WriteLine("Informe o raio do circulo: ");
            raio = Convert.ToDouble(Console.ReadLine());
            area = pi * (raio * raio); //CALCULO DA AREA
            Console.WriteLine("A area do circulo é: " + area);
            System.Threading.Thread.Sleep(5000);
```

2.3. Funções de conversão e operadores

Quando uma variável pode ser mais de um tipo, o recurso de conversão explícita pode ser usado veja:

```
float x;
double y = 10.5;
x = (float)y;
```

Estamos atribuindo um valor a variável x do tipo **float**, e esse valor é a variável **y** que por sua vez é do tipo **double** e neste caso fazendo uma conversão explícita (cast) de um tipo para o outro.

Uma outra forma de conversão de tipos é utilizar as funções de conversão da .net framework. Veja Exemplo:

```
sexo = Convert.ToChar(Console.ReadLine());
filhos = Convert.ToChar(Console.ReadLine());
salario = Convert.ToDouble(Console.ReadLine());
idade = Convert.ToInt32(Console.ReadLine());
```

Quando usamos a palavra reservada "Convert", automaticamente são apresentados os tipos de conversões que podem ser feita.

Os operadores de uma linguagem são:

- **Operadores de atribuição**
- **Operadores aritméticos**
- Operadores relacionais (trataremos desses operadores na próxima parte do curso)

- Operadores lógicos (trataremos desses operadores na próxima parte do curso)

Um operador de atribuição serve justamente para atribuirmos um valor a uma variável, veja:

```
int x;
x = 5;

//As atribuições abaixos representam o mesmo resultado, ou seja, soma
5 ao valor atual da variável.
//A segunda forma é uma forma alternativa de fazer a mesma operação
x = x + 5;
x += 5;
```

Em C# temos os seguintes operadores de atribuição:

Operador	Descrição
=	(Atribuição simples)
+=	(Atribuição aditiva)
-=	(Atribuição Subtrativa)
*=	(Atribuição Multiplicativa)
/=	(Atribuição de divisão)
%=	(Atribuição de módulo)

Os operadores aritméticos realizam operações matemáticas com as variáveis. Em C# temos os seguintes operadores aritméticos:

Operador	Descrição
+	(Adição)
-	(Subtração)
*	(Multiplicação)
/	(Divisão)
%	(Resto/Módulo)

2.4. Exercícios

Questão 01. Desenvolva os algoritmos a seguir

a) Escreva um programa que recebe três inteiros como entrada de dados e mostre na tela a soma, média e o produto destes números.

b) Fazer um programar que leia uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit.

   A fórmula de conversão é: F ← (9*C + 160) / 5, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

c) Escreva um programa que calcule e apresente o valor do volume de uma lata de óleo, utilizando a fórmula:
   VOLUME ← 3.14159 * R ^ 2 * ALTURA.

d) Construir um programa que calcule e apresente em metros por segundo o valor da velocidade de um projétil que percorre uma determinada distância em quilômetros a um determinado espaço de tempo em minutos.
   Utilize a fórmula: VELOCIDADE ← (DISTÂNCIA * 1000) / (TEMPO * 60).

e) Elaborar um programa de computador que calcule e apresente o valor do volume de uma esfera.
   Utilize a fórmula: VOLUME ←(4 / 3) * 3.14159 * (RAIO ^ 3).

f) Elaborar um programa que leia dois valores desconhecidos representados pelas variáveis A e B. Calcular e apresentar os resultados das quatro operações aritméticas básicas.

g) Escrever um programa que leia o valor correspondente ao salário mensal (variável SM) de um trabalhador e também o valor do percentual de reajuste (variável PR) a ser atribuído. Armazenar e apresentar o valor do novo salário (variável NS).

h) Escrever um programa que efetue o cálculo da quantidade de litros de combustível gasto em uma viagem, utilizando um automóvel que faz 12 km por litro. Para obter o cálculo, o usuário deve fornecer o tempo gasto e a velocidade média durante a viagem.

   Desta forma, será possível obter a distância percorrida com a fórmula: DISTÂNCIA ← TEMPO * VELOCIDADE.

   Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizado na viagem com a fórmula:
   LITROS_USADOS ← DISTANCIA / 12.

O programa deve apresentar os valores da velocidade média , tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizados na viagem.

i) Uma companhia de carros paga a seus empregados um salário de R$ 650,00 por mês mais uma comissão de R$ 50,00 para cada carro vendido e mais 5% do valor da venda.
Elabore um programa para calcular e imprimir o salário do vendedor num dado mês recebendo como dados de entrada o nome do vendedor, o número de carros vendidos e o valor total das vendas.

j) Uma conta de Energia Elétrica é calculada a partir da quantidade de Kw consumidos por mês por um cliente, acrescidos dos Impostos abaixo:

ICMS = 25%
PIS = 1%
COFINS = 4,6 %

Desenvolva um algoritmo que a partir da quantidade de Kw consumidos por mês e do valor do Kw, apresente o valor a pagar da conta de Energia Elétrica de um cliente.

3. Estruturas de decisão e de repetição

3.1. Estruturas de decisão

Normalmente, os comandos em um programa são executados um depois do outro, na seqüência em que estão escritos. Isto é chamado de execução seqüencial. Nas linguagens de programação existem vários comandos que permite ao programador especificar que o próximo comando a ser executado pode ser outro que não é o próximo da seqüência.

O C# oferece três tipos de estruturas de seleção. A estrutura de seleção **if** executa uma condição para **true (verdadeira)** ou pula a ação se a condição for **false (falsa)**. A estrutura de seleção **if/else** executa uma ação se uma condição for **true** e executa uma ação diferente se a condição for **false**. A estrutura de seleção **switch** executa uma de muitas ações diferentes, dependendo do valor de uma expressão inteira.

A estrutura de seleção **if** é uma estrutura de seleção única no qual uma ação é executada ou ignorada a partir de uma condição. A estrutura **if/else** é uma estrutura de seleção dupla – seleciona entre duas ações diferentes. A estrutura de seleção **switch** é uma estrutura de seleção múltipla, no qual uma ação é solucionada para ser executada dentre muitas ações diferentes.

Uma estrutura de seleção é usada para se escolher cursos de ação alternativos. Por exemplo, a nota para ser aprovado no IFBA tem que ser maior ou igual a 6. O comando em pseudocódigo ficaria assim:

```
Se a nota do aluno é maior ou igual a 6
Exibir "Aprovado"
```

Veja agora como ficaria a condição acima em C#:

```
if (notaAvaliacao >= 6)
{
    Console.WriteLine("Aprovado");
}
```

A estrutura de seleção **if** executa uma ação indicada só quando a condição for verdadeira; caso contrário, a ação é saltada. A estrutura de seleção **if/else** permite ao programador especificar que uma ação deve ser executada quando a condição é verdadeira e uma ação diferente quando a condição for falsa. Vejamos agora mais um comando em pseudocódigo:

```
Se a nota do aluno é maior ou igual a 6
Exibir "Aprovado"
Senão
Exibir "Reprovado"
```

Veja agora como ficaria a condição acima em C#:

```
if (notaAvaliacao >= 6)
{
    Console.WriteLine("Aprovado");
}
else
{
    Console.WriteLine("Reprovado");
}
```

Estruturas **if/else** aninhadas testam múltiplos casos colocando estruturas de seleção **if/else** dentro de outras estruturas de seleção **if/else**. Por exemplo, o seguinte comando em pseudocódigo abaixo imprimirá o conceito A para notas maiores ou iguais a 9, B para notas no intervalo de 8 a 8.9, C para notas no intervalo de 7 a 7.9, D para notas no intervalo de 6 a 6.9, e R para as demais notas:

```
Se nota do aluno for maior ou igual a 9
Exibir "A"
Senão
     Se nota do aluno for maior ou igual a 8
          Exibir "B"
Senão
Se nota do aluno for maior ou igual a 7
                Exibir "C"
     Senão
Se nota do aluno for maior ou igual a 6
                     Exibir "D"
Senão
     Exibir "R"
```

Veja agora como ficaria a condição acima em C#:

```
if (notaAvaliacao >= 9)
            {
                Console.WriteLine("A");
            }
            else if (notaAvaliacao >= 8)
            {
                Console.WriteLine("B");
            }
            else if (notaAvaliacao >= 7)
            {
                Console.WriteLine("C");
            }
            else if (notaAvaliacao >= 6)
            {
                Console.WriteLine("D");
            }
            else
            {
                Console.WriteLine("R");
            }
```

A sintaxe da condição **if/else** segue a indicação abaixo:

```
      if (CONDIÇÃO)
      {
            //Código
      }
       else if (CONDIÇÃO)
      {
           //Código
      }
      else
      {
      //Código
      }
```

Até o momento discutimos a estrutura de seleção if de seleção única e a estrutura **if / else** de seleção dupla. Um algoritmo pode conter uma série de decisões em que uma variável ou expressão será separadamente testada para cada um dos valores que ela pode assumir e ações diferentes serão executadas. A estrutura **switch** consiste em uma série de rótulos **case** e um caso **default** opcional. Veja abaixo um exemplo utilizando o **switch**. Neste exemplo, o **switch**

testará o valor do conceito de um aluno, a partir do conceito o programa exibirá a faixa de valor da nota tirada pelo aluno.

```
      switch (conceito)
             {

                 case "A":
                     Console.WriteLine("Nota entre 9 e 10");
                     break;
                 case "B":
                     Console.WriteLine("Nota entre 8 e 9");
                     break;
                 case "C":
                     Console.WriteLine("Nota entre 7 e 8");
                     break;
                 case "D":
                     Console.WriteLine("Nota entre 6 e 7");
                     break;
                 default:
                     Console.WriteLine("Nota inferior a 6");
                     break;

             }
```

Vejamos abaixo a sintaxe do comando **switch**

```
      switch(VARIAVEL)
      {
          case VALOR:
                  //Faz algo se VARIAVEL for igual ao VALOR
                  break;
          default:    //(OPCIONAL)
                  //Faz algo se VARIAVEL não for igual a nenhum
CASE
                  break;
      }
```

3.2. Operadores Relacionais e Operadores Lógicos

Em C# temos os seguintes operadores relacionais:

Operador	Descrição
==	(Igualdade)
>	(Maior)

<	(Menor)
<=	(Menor igual)
>=	(Maior igual)
!=	(Diferente)

Existe um tipo de erro bastante comum entre os programadores da linguagem C que é trocar acidentalmente os operadores de igualdade (==) e atribuição (=). Comandos com esse erro compilam em geral compilam corretamente, o que provavelmente irá gerar resultados incorretos através de erro de lógica durante a execução.

A seguir temos um exemplo que utiliza estes operadores:

```
            int valor = 5;

            if (valor > 100)
            {
                Console.WriteLine("Valor superior a 100.");
            }
            else if (valor < 100)
            {
                if (valor >= 50)
                {
                    Console.WriteLine("Valor menor que 100 e maior ou
igual a 50.");
                }
                else if (valor < 50)
                {
                    if (valor != 10)
                    {
                        if (valor == 5)
                        {
                            Console.WriteLine("Valor igual a 5.");
                        }
                    }
                }

            }
```

É possível avaliar diversos **ifs/elses** com uma determinada expressão. Para isso utilizamos os operadores lógicos. Os operadores lógicos são usados quando precisamos testar mais de uma expressão condicional.

Veja na tabela abaixo a relação com os operadores lógicos em C#:

Operador	Descrição

&& (E)	Se duas expressões condicionais forem verdadeiras o resultado é verdadeiro.
| | (OU)	Se qualquer expressão condicional for verdadeira o resultado é verdadeiro.
! (NEGAÇÃO)	Se a expressão condicional for falsa o resultado é verdadeiro. Se a expressão for verdadeira o resultado é falso.

Algumas expressões complexas podem exigir mais de um operador lógico. A sequência de execução neste tipo de expressão é primeira, os operadores matemáticos, depois os operadores de comparação e só então os operadores lógicos.

Abaixo apresentamos um exemplo de como utilizar os operadores lógicos.

```
static void Main(string[] args)
        {
            string veiculo;
            char pintura;

            Console.WriteLine("FORD VEICULOS \n");
            Console.WriteLine("Informe o veiculo: (Ford Ka, Fiesta ,
Focus, Ranger, Fusion)");
            veiculo = Console.ReadLine();
            Console.WriteLine("Informe a pintura do veiculo: (s -
solida, m - metalica , p - perolizada)");
            pintura = Convert.ToChar(Console.ReadLine());



            if (veiculo != "Ford Ka")
            {

                if (veiculo == "Focus" || veiculo == "Fusion")
                {
                    Console.WriteLine("Itens de serie: ArCondicionado,
Direção Hidraulica, Vidros e Travas eletricias");
                }
                else
                {
                    if (veiculo == "Ranger" && pintura == 's')
                    {
                        Console.WriteLine("Voce tem direito a escolher
uma pintura metalica gratis!");

                    }
                    else if (veiculo == "Fiesta" && (pintura == 'm' ||
pintura == 's'))
```

```
                    {
                        Console.WriteLine("Itens de serie: Aerofolio,
Freios Abs");

                    }
                }

            }
            else
            {
                Console.WriteLine("Itens de serie: Alarme e Trava
Eletrica");
            }
            System.Threading.Thread.Sleep(5000);
        }
```

3.3. Estruturas de repetição

As estruturas de repetição permitem executar uma linha ou bloco de código repetidamente até que uma condição seja verdadeira. O uso de laços na programação é uma prática comum, pois em muitos casos é preciso percorrer uma determinada coleção de dados, um conjunto de registros, valores de vetores e matrizes, etc.

Existem 4 estruturas de repetição no C#, são elas:

a) while
b) do... while
c) for
d) foreach

While

A estrutura **while** é executada sempre associado a uma condição, ou seja, a cada passagem pelo looping a condição é avaliada. A estrutura **While** é bem mais simples que as demais estruturas, pois, nele não é necessário declarar uma variável de inicialização, nem mesmo o incremento, somente é necessário declarar a condição de parada da repetição. Apesar de você não precisar declarar o incremento dentro da estrutura **while**, você tem que preocupar com a condição para que seu trecho de código saia do laço, pois, caso a condição não seja falso em algum momento, você irá criar um loop infinito, fazendo com que o código presente dentro do laço **while** nunca pare de ser executado.

Veja como fica a sintaxe do comando **while**:

```
while (CONDIÇÃO)
{
//Código
}
```

A condição expressa na estrutura **while** é uma expressão booleana, ou seja, sempre retorna falso ou verdadeiro e a instruções dentro do bloco de código só será executada enquanto essa expressão retornar verdadeiro. Veja um exemplo:

```
class Program
    {
        static void Main(string[] args)
        {
            int contador = 0;

            while (contador < 5)
            {
                Console.WriteLine("Regilan Meira Silva");
                contador++;
            }
            System.Threading.Thread.Sleep(5000);

        }
    }
```

No exemplo acima temos uma variável chamada contador e seu valor inicial é 0, na CONDIÇÃO do bloco while testamos se a variável contador é menor que 5, caso verdadeiro mostramos na tela o nome "Regilan Meira Silva" e o incrementos em 1 e o código dentro do bloco while continuará até que essa condição se torne falsa.

Do ... While

A estrutura **do ... while** é bastante semelhante à estrutura **while**, tendo como principal diferença que a condição da estrutura é testada no final e não no inicio como acontece no **while**. No **while** existe a possibilidade do código não ser executado nenhuma vez, pois está vinculado ao resultado da condição que é testada no início do bloco. No **do ... while** o código escrito dentro da estrutura será executado pelo menos uma vez, isso porque a verificação é realizada apenas no final da estrutura.

Veja como fica a sintaxe da estrutura **do ... while**:

```
do
{
// Código
}
while (CONDIÇÃO)
```

Agora veremos o mesmo exemplo da estrutura **while**, só que agora usaremos o **do ... while**

```
int contador = 0;

do
{
   Console.WriteLine("Regilan Meira Silva");
   contador++;
} while (contador < 5);
System.Threading.Thread.Sleep(5000);
```

For

Toda estrutura **For** é constituída de uma variável de inicialização, uma condição e o incremento. O laço **for** trabalha checando uma condição para executar um bloco de código até que essa condição seja verdadeira. Conforme foi citado no inicio do parágrafo, no laço **for** temos que em sua sintaxe declarar sua inicialização, sua condição e seu incremento, veja no exemplo abaixo como fica a sintaxe da estrutura **for:**

```
for (int i =0; i <= 5; i++)
{
      //Código              Incremento
}
            Condição
        Inicialização
```

Explicando o código acima, na primeira parte declaramos uma variável do tipo inteiro (**int**) e a inicializamos com o valor 0 (zero), na segunda parte temos a condição nesse caso verifica se a nossa variável recém criada é menor ou igual a 5 e a terceira e ultima parte é o incremento desta variável, sendo essas três partes separadas por ';' (ponto e virgula). Um incremento nada mais é do que adicionar 1 a uma variável, ou seja, se uma variável vale 0 (zero) e passa

por um incremento logo essa variável vale 1 e se passa mais uma vez vale 2 e assim por diante.

Como vemos o funcionamento é bastante simples, todo o código dentro desse bloco do 'for' será executado 5 vezes.

For Each

A estrutura **foreach** é utilizada para percorrer coleções ou **arrays**. Sua estrutura é formada pelo tipo de dado, o nome da variável e a lista a ser percorrida. No próximo capitulo iremos aprender sobre **arrays** no próximo, mas para mostrar o funcionamento do laço **foreach** utilizaremos um **array** como exemplo.

Veja a sintaxe básica do laço foreach:

```
foreach(<tipo de dado> <nome> in <lista>)
{
        //Código
}
```

Para facilitar o entendimento, vamos ver o seguinte exemplo no C#. Neste exemplo criamos um **array** de string e colocamos alguns elementos dentro e no nosso laço **foreach** será exibida todos os elementos dentro de nosso **array**.

```
class Program
    {
        static void Main(string[] args)
        {
            // array de strings = declaração e inicialização do array
            string[] nomes = { "Regilan", "Aline", "Jose", "Maria",
"João"};

            foreach (string pessoa in nomes)
            {
                Console.WriteLine(pessoa);
            }

            System.Threading.Thread.Sleep(5000);
```

```
        }

    }
```

3.4. Exercícios

Questão 01. Elabore um programa em C# que verifique se um dado número inteiro positivo é PAR ou IMPAR.

Questão 02. Faça um programa que leia dois valores numéricos e um símbolo. Caso o símbolo seja um dos relacionados abaixo efetue a operação correspondente com os valores. Atenção para a divisão por zero!

- "+" operação de soma
- "-" operação de subtração
- "*" operação de multiplicação
- "/" operação de divisão

Questão 03. Dados 3 (três) valores A, B, C, verifique se podem ser valores de lados de um triângulo (cada lado ser menor do que a soma dos outros dois) e, se for, determine se o triângulo é equilátero, isósceles ou escaleno.

Questão 04. A prefeitura de Ilhéus abriu uma linha de crédito para seus servidores. O valor máximo da prestação não poderá ultrapassar 30% do salário bruto. Através do salário bruto e do valor da prestação, informar se o empréstimo pode ou não ser concedido.

Questão 05. Construir um programa para calcular as raízes de uma equação do 2º grau, sendo que os valores dos coeficientes A, B, e C devem ser fornecidos pelo usuário através do teclado.

Questão 06. Criar um programa que a partir da idade e peso do paciente calcule a dosagem de determinado medicamento e imprima a receita informando quantas gotas do medicamento o paciente deve tomar por dose. Considere que o medicamento em questão possui 500 mg por ml, e que cada ml corresponde a 20 gotas.

a) Adultos ou adolescentes desde 12 anos, inclusive, se tiverem peso igual ou acima de 60 quilos devem tomar 1000 mg; com peso abaixo de 60 quilos devem tomar 875 mg.
b) Para crianças e adolescentes abaixo de 12 anos a dosagem é calculada pelo peso corpóreo conforme a tabela a seguir:

Peso	Dosagem
5 kg a 9 kg	125 mg
9.1 kg a 16 kg	250 mg
16.1 kg a 24 kg	375 mg
24.1 kg a 30 kg	500 mg
Acima de 30 kg	750 mg

Questão 07. Criar um programa que informe a quantidade total de calorias de uma refeição a partir do usuário que deverá informar o prato, a sobremesa e a bebida (veja a tabela a seguir).

Prato	Calorias	Sobremesa	Calorias	Bebida	Calorias
Vegetariano	180 cal	Abacaxi	75 cal	Chá	20 cal
Peixe	230 cal	Sorvete *diet*	110 cal	Suco de laranja	70 cal
Frango	250 cal	Mouse *diet*	170 cal	Suco de melão	100 cal
Carne	350 cal	Mouse chocolate	200 cal	Refrigerante *diet*	65 cal

Questão 08. Desenvolva um programa que recebe do usuário o placar de um jogo de futebol (os gols de cada time) e informa se o resultado foi um empate, a vitória do primeiro time ou do segundo time.

Questão 09. Pelas regras da escola ABC , toda disciplina é composta de 3 avaliações obrigatórias. Se um aluno tem média superior ou igual a 7 (sete) nas três avaliações, ele é considerado "Aprovado por média.". Se tiver média inferior a 7 (sete), tem direito a uma quarta prova, chamada de reposição, que substituiu a menor nota das três provas anteriores (se a nota da reposição for maior que ela). Após a reposição, se as três maiores notas formarem uma média igual ou superior a 7 (sete), o aluno é considerado "Aprovado na reposição.". Finalmente, apenas se tiver a média com a reposição inferior a 7 (sete), ele faz uma quinta prova, chamada de final, em que é considerado "Aprovado na final." se a soma da média com a reposição e a nota da prova final for igual ou maior que 12 (doze). Nesse último caso, caso a soma da média com reposição e prova final for inferior a 12 (doze), o aluno é dito como "Reprovado.". Faça um programa que leem as três notas

obrigatórias de uma disciplina e depois, apenas se necessário, lê as notas da reposição e final. O programa deve dar como saída às frases "Aprovadas por média.", "Aprovado na reposição". "Aprovado na final." ou "Reprovado." respectivamente em cada uma dessas situações conforme a descrição anterior.

Questão 10. Faça um programa que requisita do usuário 6 (seis) números inteiros para apostar na megasena. O programa também deverá receber os seis números sorteados. O programa deve informar quantos pontos o usuário acertou.

Questão 11. Um aluno de computação está organizando um bolão de futebol. Segundo suas regras, os apostadores informam o placar do jogo e ganham 10 pontos se acertarem o vencedor ou se foi empate e ganham mais 5 pontos para o placar de cada time que acertarem. Exemplo: se o placar do jogo foi 3x2, são 0 pontos se o placar apostado foi 0x1; 5 pontos para os placares apostados 0x2 ou 3x5; 10 pontos para o placar apostado 1x0; ou 20 pontos para o placar exato de 3x2. Faça um programa que requisita do usuário o placar apostado e depois o placar do jogo e informa quantos pontos o apostador fez.

Questão 12. Desenvolva um programa em C# que determinará se um cliente de uma loja de departamentos excedeu o limite de crédito em conta corrente. Para cada cliente, os seguintes fatos estão disponíveis:

a) Número da conta (um inteiro)
b) Saldo no princípio do mês
c) Total de todos os itens comprados a crédito por este cliente neste mês
d) Total de todos os créditos do cliente nesta conta neste mês
e) Limite de crédito permitido

Questão 13. Desenvolver um programa em C# que efetue a soma de todos os números ímpares que são múltiplos de três e que se encontram no conjunto dos números de 1 até 500.

Questão 14. Escreva um programa que gere os números de 1000 a 1999 e escreva somente os números que são divisíveis por 11 e cujo resto é 5.

Questão 15. Faça um programa que leia 2 valores inteiros e positivos: X e Y. O algoritmo estruturado deve calcular e escrever a função potência x Y.

Questão 16. Faça um programa que leia um numero, calcule seu fatorial e mostre o resultado obtido.

Questão 17. Escreva um algoritmo que lê um valor n inteiro e positivo e que calcula a seguinte soma: S := 1 + 1/2 + 1/3 + 1/4 + ... + 1/n.

Questão 18. Escreva um programa em C# que converta galões para litros. O programa deve mostrar valores de galão entre 10 a 20 em incrementos de um, juntamente com o valor equivalente em litros. Use a relação de 1 galão para 3,78 litros.

Questão 19. Uma máquina comprada por R$ 28,00 se deprecia com uma taxa de R$ 4,00 por ano em sete anos. Escreva um programa em C# que calcula e mostra a tabela de depreciação para sete anos.

A tabela deve ter a forma abaixo:

Ano	Depreciação	Valor no fim do ano	Depreciação Acumulada
1	4000	24000	4000
2	4000	20000	8000
3	4000	16000	12000
4	4000	12000	16000
5	4000	8000	20000
6	4000	4000	24000
7	4000	0	28000

Questão 20. Escreva um programa em C# para encontrar a soma, a soma dos quadrados e a soma dos cubos dos primeiros **n** inteiros, começando com **1** e terminando com **n = 100** Verifique se em cada caso as igualdades se abaixo confirmam.

$$1 + 2 + 3 + \ldots + n = n(n+1)/2$$
$$1^2 + 2^2 + 3^2 + \ldots + n^2 = n(n+1)(2n+1)/6$$
$$1^3 + 2^3 + 3^3 + \ldots + n^2 = n^2(n+1)^2/4$$

Questão 21. Uma rainha requisitou os serviços de um monge e disse-lhe que pagaria qualquer preço. O monge, necessitando de alimentos, indagou à rainha sobre o pagamento, se poderia ser feito com grãos de trigo dispostos em um tabuleiro de damas, de tal forma que o primeiro quadro deveria conter apenas um grão e os quadros subsequentes, o dobro do quadro anterior. A rainha achou o trabalho barato e pediu que o serviço fosse executado, sem se dar conta de que seria impossível efetuar o pagamento. Faça um algoritmo para calcular o número de grãos que o monge esperava receber. A imagem ao lado representa um tabuleiro de damas.

Questão 22. Supondo que a população de um país A seja da ordem de 90.000.000 de habitantes com uma taxa anual de crescimento de 3% e que a população de um país B seja, aproximadamente, de 200.000.000 de habitantes com uma taxa anual de crescimento de 1.5%. Escreva um programa em que calcule o número de anos necessário para que a população do país A ultrapasse ou iguale a população do país B mantido essas taxas de crescimento.

Questão 23. Foi feita uma pesquisa entre os habitantes de uma região. Foram coletados os dados de idade, sexo (M/F) e salário. Faça um programa que calcule e mostre:

- A média dos salários do grupo;
- A maior e a menor idade do grupo;
- A quantidade de mulheres com salário até R$ 200,00;
- A idade e o sexo da pessoa que possui o menor salário;

Finalize a entrada de dados ao ser digitada uma idade negativa.

Questão 244 Foi realizada uma pesquisa de algumas características físicas da população de uma certa região, a qual coletou os seguintes dados referentes a cada

habitante para serem analisados: sexo (F ou M), cor dos olhos (A - Azuis, V - Verdes, P - pretos ou C - Castanhos), cor dos cabelos (L - loros, C - Castanhos, P - Pretos) e idade.

Faça um algorítmo que leia os dados enquanto o usuário desejar. Para isso, mostre uma mensagem: "Deseja continuar (S/N)?". O usuário deverá responder S ou N. Ao final Imprima os seguintes resultados:

- A menor idade.
- A quantidade de mulheres cuja idade está entre 18 e 35 anos, incluindo estes valores.
- A quantidade de homens de cabelos castanhos ou que tenham idade maior que 28 anos.
- A média de idade das mulheres de olhos castanhos.

4. Arrays e Funções

4.1. Arrays

Os arrays são estruturas de dados que consistem em itens de dados do mesmo tipo, relacionados entre si. Um array é um grupo de posições de memória consecutivas, todas de mesmo nome e mesmo tipo. Para fazer referência a uma posição particular ou elemento no array, especificamos o nome do array e o número da posição do elemento no array. Em C# o primeiro elemento em um array é o elemento zero. Dessa forma, o primeiro elemento do array é referenciado como: **exemploArray[0]**, o segundo elemento é representando como **exemploArray[1]** e assim sucessivamente.

Os arrays ocupam espaço na memória, dessa forma o programador deverá especificar o tipo de cada elemento e o número de elementos exigidos pelo array. Pode-se reservar memória para vários arrays com uma única declaração. Para se declarar um array, use de um colchete [] antes do nome da variável. Exemplo:

```
string[] arrayDeString;
```

O tipo array pode ter várias dimensões, o tamanho dele é definido pelo desenvolvedor, mas é importante saber que o primeiro índice é sempre zero. Portanto, um array declarado com dez posições, na realidade terá onze posições: 0 a 10.

Vamos ver alguns exemplos de declaração de array.

```
class Program
{
        static void Main(string[] args)
        {
            string[] alunos = new string[100]; //declara e reserva 100
elementos do array de string alunos
            double[] notas = new double[100];//declara e reserva 100
elementos do array de string alunos
        }
}
```

No exemplo abaixo temos um programa para obter o nome e a nota de um aluno. Para isso foi reservado dois arrays: o primeiro para armazenar o nome do

aluno e o segundo para guardar a sua nota. Para preencher os valores no array, estamos utilizando a estrutura (***for***) para percorrer cada elemento e armazenar o valor. Veja o exemplo completo:

```
class Program
{
        static void Main(string[] args)
        {
            string[] alunos = new string[3];
            double[] notas = new double[3

            Console.WriteLine("PROGRAMA PARA OBTER AS NOTAS DOS
ALUNOS\n");
            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine("Informe o nome do aluno: ");
                alunos[i] = Console.ReadLine();
                Console.WriteLine("Informe a nota do aluno: ");
                notas[i] = Convert.ToDouble(Console.ReadLine());
            }
        }
}
```

Um array pode ser declarado com várias dimensões, por exemplo uma matriz contendo dados. Para ser declarado um array de várias dimensões, deve-se declarar colocando vírgulas dentro dos colchetes. Veja exemplo:

```
     string[] arrayDeString1; //array de string de uma dimensão
     string[,] arrayDeString2; //array de string de duas dimensões
     string[, ,] arrayDeString3; //array de string de três dimensões
     string[, , ,] arrayDeString4; //array de string de quatro
dimensões
```

Na imagem abaixo temos um array com duas dimensões. O array contém três linhas e quatro colunas, por isso diz-se que ele é um array 3 por 4. Cada elemento do array é identificado por um nome na forma **a[i][j]; a é o nome do array, i e j identificam um elemento específico no array a.**

a[0][0]	**a[0][1]**	**a[0][2]**	**a[0][3]**
a[1][0]	**a[1][1]**	**a[1][2]**	**a[1][3]**
a[2][0]	**a[2][1]**	**a[2][2]**	**a[2][3]**

Veja agora um exemplo utilizando arrays bidimensionais (matrizes):

```
string[,] aluno = new string[2,3];

aluno[0,0] = "Eduardo";
aluno[0,1] = "Ane";
aluno[0,2] = "Hyago";

aluno[1,0] = "Fabrini";
aluno[1,1] = "Johacia";
aluno[1,2] = "Pedro Paulo";
```

4.2. Funções

A maioria dos programas de computador são muito maiores dos que os exemplos já feito nessa série de estudo sobre C#. A melhor maneira de desenvolver e manter um programa grande é construí-lo a partir de partes menores, que são conhecidas como **funções**.

Além das funções construídas pelos programadores, a biblioteca do .Net Framework oferece uma rica coleção de funções já prontas para cálculos matemáticos, operações com strings, entradas e saídas de dados, entre outras. Vamos ver um exemplo do uso de uma função do .Net Framework.

```
int numero = 2;
int novonumero;
novonumero = Math.Pow(numero, 3);
```

Quando na última linha de código utilizamos a função **Math.Pow**, estamos nos referindo a função **POW** da biblioteca **MATH**. A função **POW** eleva um número a uma determinada potência. Essas funções presentes na plataforma facilitam o trabalho do programador porque estas funções oferecem muitos dos recursos de que os programadores precisam. Recomendo que faça uma pesquisa e familiarize-se com a rica coleção de funções e classes da .Net Framework. Evite reinventar a roda, utilize as funções da biblioteca ao invés de escrever funções novas.

Uma função que é amplamente utilizada em nossos códigos fontes, que é as funções de CONVERSÃO DE TIPOS. Essas funções estão presentes na classe **System.Convert**. A tabela abaixo apresenta algumas dessas funções:

Função	Objetivo
Convert.ToBoolean	Converte para um valor booleano
Convert.ToChar	Converte para um caractere unicode
Convert.ToDateTime	Converte para um valor do tipo datetime
Convert.ToDecimal	Converte para um decimal
Convert.ToDouble	Converte para um número de ponto flutuante de precisão dupla
Convert.ToInt32	Converte para um inteiro de 32 bits
Convert.ToString	Converte para uma representação em formato string

As funções permitem ao programador modularizar um programa. Todas as variáveis declaradas em definições de função são variáveis locais, ou seja, elas só são conhecidas na função na qual são definidas. A maioria das funções possui uma lista de parâmetros que são responsáveis por prover os meios para transferir informações entre funções. Os parâmetros de uma função também são variáveis locais. O formato da chamada da função inclui a especificação do nome colocando os argumentos dos parâmetros entre parênteses.

Como já sabemos, uma função é um bloco de instruções que é executado quando é chamada de algum outro ponto do programa. Seu formato é o seguinte:

tipo nome (parametro1, parametro 2, ...) conteúdo

- **tipo:** é o tipo de dados que a função retorna.
- **nome:** é o nome pelo qual será possível chamar a função.
- **parâmetro:** (podem ser especificados quantos você quiser). Cada argumento consiste em um tipo de dados seguido pelo seu identificador, como na declaração de uma variável (por exemplo, int x) e que funciona dentro da função como qualquer outra variável. Eles permitem a passagem de parâmetros para a função quando é chamada. Os parâmetros diferentes são separados por vírgulas.

- **conteúdo:** é o corpo da função. Pode ser uma única instrução ou um bloco de instruções. No último caso, precisa ser delimitado por chaves {}.

Considere o exemplo abaixo onde temos a definição da função quadradoDeUmNumero. A função retorna um inteiro e possui um parâmetro onde é informado o número que será elavado ao quadrado.

```
int quadrado(int numero)
{                    Nome da função
    return numero * numero; // Retorno
}
```

Tipo do retorno

Parâmetros da função

```
        static void Main(string[] args)
        {
            int n;
            Console.WriteLine("Informe um numero: ");
            n = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("O quadrado do numero é: " + quadrado(n)
); //Executa a função
            System.Threading.Thread.Sleep(5000);


        }

        //Como o método void main é estático, a função quadrado também
deverá ser estático
        static int quadrado(int numero)
        {
            return numero * numero;
        }
```

Vamos agora lembrar da sintaxe de uma declaração de função: **tipo nome (parametro1, parametro2 ...) conteúdo**. Estudamos que é obrigatório que a declaração comece com um tipo, que é o tipo de dados que será retornado pela função com a instrução **return**. Acontece que no C# podemos escrever funções sem retorno de valor. Imagine que necessitamos fazer uma função somente para exibir uma mensagem na tela. Nós não precisamos que retorne nenhum valor, e mais ainda, não precisamos receber nenhum parâmetro. Para casos assim podemos utilizar uma função com retorno **void**.

Mais um exemplo:

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

     imprimirNaTela(); // chamando a função

}

//define uma função para imprimir algo na tela
 static void imprimirNaTela()
 {
     Console.WriteLine("EU SOU UMA FUNÇÃO DO TIPO VOID");
     System.Threading.Thread.Sleep(5000);

 }
```

4.3. Exercícios

Questão 01. O máximo divisor comum de dois inteiros é o maior inteiro que divde cada um dos dois números. Escreva uma função MDC que retorna o máximo divisor comum de dois inteiros.

Questão 02. Escreva uma função que receba a média de um aluno e retorne 4, se ela estiver no intervalo 90-100, 3 se a média estiver no intervalo 80-89, 2 se a média estiver no intervalo 70-79, 1 se a média estiver no intervalo 60-69, e 0 se a média for menor que 60.

Questão 03. Faça uma função que recebe, por parâmetro, a altura (alt) e o sexo de uma pessoa e retorna o seu peso ideal. Para homens, calcular o peso ideal usando a fórmula peso ideal = 72.7 x alt - 58 e, para mulheres, peso ideal = 62.1 x alt - 44.7.

Questão 04. Escreva um procedimento que recebes 3 valores reais X, Y e Z e que verifique se esses valores podem ser os comprimentos dos lados de um triângulo e, neste caso, retornar qual o tipo de triângulo formado. Para que X, Y e Z formem um triângulo é necessário que a seguinte propriedade seja satisfeita: o comprimento de cada lado de um triângulo é menor do que a soma do comprimento dos outros dois lados. O procedimento deve identificar o tipo de triângulo formado observando as seguintes definições:

- Triângulo Equilátero: os comprimentos dos 3 lados são iguais.
- Triângulo Isósceles: os comprimentos de 2 lados são iguais.
- Triângulo Escaleno: os comprimentos dos 3 lados são diferentes.

Questão 05. A prefeitura de uma cidade fez uma pesquisa entre os seus habitantes, coletando dados sobre o salário e número de filhos. Faça um procedimento que leia esses dados para um número não determinado de pessoas e retorne a média de salário da população, a média do número de filhos, o maior salário e o percentual de pessoas com salário até R$350,00.

Questão 06. Faça um procedimento que recebe, por parâmetro, um valor N e calcula e escreve a taboada de 1 até N. Mostre a tabuada na seguinte forma abaixo:
1 x N = N

2 x N = 2N

...

N x N = N2

Questão 07. Deseja-se publicar o número de acertos de cada aluno em uma prova em forma de testes. A prova consta de 30 questões, cada uma com cinco alternativas identificadas por A, B, C, D e E. Para isso são dados:

- o cartão gabarito;
- o número de alunos da turma;
- o cartão de respostas para cada aluno, contendo o seu número e suas respostas

Questão 08. Dados dois strings (um contendo uma frase e outro contendo uma palavra), determine o número de vezes que a palavra ocorre na frase.

Exemplo:

Para a palavra ANA e a frase :

ANA E MARI**ANA** GOSTAM DE B**ANANA** (2)

Temos que a palavra ocorre 4 vezes na frase.

Questão 09. Faça uma função que obtenha do teclado o valor n e imprima na tela os n primeiros termos de uma seqüência de Fibonacci.
Observação: Uma seqüência é dita de Fibonacci quando os dois primeiros termos forem 0 e 1 e todos os termos consecutivos tenham valor igual à soma dos dois termos anteriores.

Exemplo: 0 – 1 – 1 – 2 – 3 – 5 – 8 – 13 – 21 – 34 – 55 .

Questão 10. Faça um algoritmo que receba uma matriz A[5][5] e realize as seguintes trocas:

a) linha 2 com a coluna 4;

b) coluna 2 com a diagonal principal ;

c) diagonal principal com a secundária;

d) linha 5 com a linha 1;

Após realizar as trocas apresente a nova matriz.

5. Enumeradores, Modificadores de Acesso e Structs

5.1. Enumeradores

Enumeradores são tipos que possuem valores fixos. São geralmente usados para que o desenvolvedor use uma lista de opções possíveis em um programa e para substituirmos constantes nomeadas que são relacionadas. Vamos supor um exemplo, suponha que você precisa definir uma variável cujo valor irá representar um dia da semana. Há apenas sete valores significativos que essa variável sempre armazenará. Para definir esses valores, você pode usar um tipo de enumeração. Outro exemplo de enumeração: definir valores para representar os meses de um ano.

Para criar um enumerador a sintaxe é representada como a do exemplo a seguir:

```
      enum
DiasDaSemana{Domingo,Segunda,Terca,Quarta,Quinta,Sexta,Sabado};
```

Por padrão, os enumeradores criados sem um tipo definido de dado, são interpretados como sendo do tipo inteiro (int / Integer).

Os enumeradores podem ainda suportar os tipos (byte / Byte, sbyte / SByte, short / Short, ushort / UShort, int / Integer, uint / UInteger, long Long, ulong / ULong).

Veja outro exemplo abaixo onde usamos os enumeradores para representar os tipos de usuários de um sistema. Observe que esses enumeradores são do tipo ***byte***.

```
   public enum TipoDoUsuario : byte
        {
            Administrador= 0,
            Gerente = 1,
            Supervisor = 2,
            UsuarioComum = 3
        }
```

Dentre todas as vantagens para utilizar os enumeradores, a principal delas é a capacidade de deixar um código muito limpo e fácil de ser compreendido, contribuindo para uma padronização no desenvolvimento. Veja na imagem abaixo a utilização de um enumerador.

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string nome = "Regilan Meira";
            int idade = 25;
            byte tipoUsuario = TipoDoUsuario.
        }

        public enum TipoDoUsuario : byte
        {
            Leitor = 0,
            Contribuidor = 1,
            Revisor = 2,
            Administrador = 3

        }
```

Administrador
Contribuidor
Leitor
Revisor

Observe que ao usarmos o enumerador TipoDoUsuario, o Intellissense mostrará automaticamente os tipos definidos no enumerador TipoDoUsuario. Isso facilita o desenvolvimento e a leitura do código fonte. Conclusão: É muito mais simples definir um enumerador do que sair criando diversas constantes no código para definir um tipo fixo.

5.2. Modificadores de Acesso

Até o momento criamos tipos e variáveis sem se preocupar com a visibilidade desses objetos em um projeto, ou seja, não atentamos para o fato sobre quais objetos poderão acessá-las. Este tipo de controle é feito através dos modificadores de acessos. O modificador de acesso diz qual é o alcance que essa variável terá, a tabela abaixo mostra quais são os possíveis modificadores e seu alcance, quando esse valor não é digitado o padrão(default) usado é private.

Operador	Descrição

Private	Acesso limitado aos membros da classe. Não pode ser acessado fora do bloco da classe e nem pelas classes derivadas. <default>
Public	Acesso ilimitado. É possível acessar mesmo fora do bloco da classe e das classes derivadas da mesma.
Protected	Acesso limitado aos membros da classe. Pode ser acessado pelas classes derivadas.
Internal	Acesso limitado ao Assembly que o contém. Pode ser acessado pelos objetos derivados da classe.
Protected Internal	Ele permite que seja usado o modificador Protected ou Internal. Sendo sua utilização baseada no contexto.

Embora existam vários tipos de modificadores, os mais usados sãos "private" e "public", sendo o primeiro usado normalmente para a declaração de elementos (atributos ou métodos de uma classe) locais e o segundo para elementos que devem ser acessíveis fora do contexto da ***struct*** ou ***classe***. O conceito dos modificadores de acesso ficará mais claro quando começarmos a estudar a programação orientada a objetos.

5.3. Structs

Uma estrutura (structs) nada mais é que um tipo de dado definido pelo usuário. Se você tem uma necessidade de se criar um objeto que possui atributos e operações a serem realizadas em torno desse objeto, criar uma estrutura é uma solução para resolver esse problema, mas é bom lembrar que quando estudarmos a orientação a objetos é mais interessante definir classes para criar objetos, ao invés de utilizar estruturas para resolver esse tipo de problema. Enquanto não estudamos a POO, vamos construir uma aplicação exemplo para a utilização de structs. Nesse programa criaremos uma estrutura chamada de **Pessoa** que possui os seguintes atributos: *nome*, *idade*, *peso* e *altura*. Além dos atributos, a estrutura possuirá um método chamado *CalculaIMC* que é responsável por calcular o índice de massa corporal da pessoa.

Veja no código abaixo a implementação dessa estrutura.

```
        struct Pessoa {
            public string Nome;
            public int Idade;
            public double Peso;
            public double Altura;

            public double CalculaIMC()
            {
                return Peso / (Altura * Altura);
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("PROGRAMA PARA CALCULAR O IMC DE UMA
PESSOA\n");

            Pessoa p;

            Console.WriteLine("Informe o nome da pessoa:");
            p.Nome = Console.ReadLine();
            Console.WriteLine("\nInforme a idade da pessoa :");
            p.Idade = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("\nInforme o peso da pessoa :");
            p.Peso = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("\nInforme a altura da pessoa :");
            p.Altura = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("\nVeja abaixo os dados dessa pessoa:\n
Nome: "
            + p.Nome + "\n Idade:" + p.Idade + "\nIMC: " +
p.CalculaIMC() );

            System.Threading.Thread.Sleep(5000);
        }
```

No código apresentado, temos a implementação da estrutura chamada de Pessoa (`struct Pessoa`). Logo em seguida, nas quatro linhas abaixo temos a declaração dos atributos (Observe que todos eles estão com o modificador de acesso `public`, dessa forma os valores são acessados na própria estrutura ou fora dela. Experimente alterar o modificador para private e tente acessar o seu valor fora da estrutura.). Depois da declaração dos atributos temos a implementação do método CalculaIMC que retorna um valor double de acordo com a fórmula do IMC.

No segundo bloco de código (`Main`) temos um programa para entrar com os dados e armazená-los na struct Pessoa. A cada linha de execução um valor será armazenado em cada um dos atributos da estrutura. No final, será exibido um resumo com os dados dessa pessoa e o índice de massa corporal calculado a partir do peso e altura. Mais uma vez recomendo alterar o modificador de acesso de um

dos atributos para private. Em seguida, declare uma nova estrutura Pessoa e verifique se o atributo que foi declarado como Private está disponível.

5.4. Exercícios

Questão 01. Escreva uma estrutura chamada Quadrado com um atributo lado do tipo double. Essa estrutura deverá possuir dois métodos que devem retornar a área e o perímetro do quadrado segundo a fórmula abaixo:

- Area = lado²
- Perimetro = 4 * lado

Questão 02. As informações a seguir referem-se a uma conta de luz. Para cada conta de luz cadastra-se: data em que a leitura do relógio de luz foi realizada, número da leitura, quantidade de Kw gasto do mês, valor do Kw, data de vencimento, data do pagamento.

Exercício: Identifique os atributos e construa uma struct para armazenar os dados acima. Nessa struct deverá também existir um método que calcula o valor a ser pago pelo cliente que é calculado da seguinte forma:

- Se o cliente pagar a conta em dias, o valor a ser pago é a quantidade de Kw gasto no mês multiplicado pelo valor do Kw.
- Se o cliente pagar a conta atrasado, o valor a ser pago é a quantidade de Kw gasto no mês multiplicado pelo valor do Kw, acrescentado um juros de 2% do valor da conta para cada dia em atraso.

Depois da struct pronta, faça um programa para ler os dados e exibir na tela o valor a ser pago pelo cliente.

Questão 03. As informações a seguir se referem a uma aplicação de controle pessoal de horário de remédios, existente no IPhone de Regilan. Para cada remédio cadastra-se: o nome de quem vai tomar o remédio, a data de início, a quantidade de dias que foi prescrita pelo médico, a quantidade de vezes ao dia, a dosagem e o nome do remédio. Ao cadastrar o remédio, a aplicação sugere todos os horários

possíveis para tomar o remédio. O usuário escolhe o melhor horário e a aplicação avisa até quando o remédio deve ser tomado e exibe na tela do IPhone os horários.

Exercício: Identifique os atributos e métodos desse cenário e crie uma struct para resolver o problema proposto acima.

Questão 04. As informações abaixo se referem a algumas aplicações bancárias existentes no Banco ABC. O banco ABC é um banco de investimentos e possui algumas aplicações onde cada uma possui uma taxa de juros diferente, conforme a tabela abaixo.

Obs.: a/m significa ao mês.

- Poupança (Pode-se resgatar o valor aplicado a qualquer momento) – 0,78 a/m
- Titulo de Capitalização (Pode-se resgatar após 24 meses) – 0,97 a/m
- CDB (Pode-se resgatar após 6 meses) – 0,85 a/m
- RDB (Pode-se resgatar a metade aplicada em seis meses e após 12 meses poderá ser resgatado a qualquer momento) – 0,90 a/m

O cliente deverá escolher uma aplicação, o valor inicial aplicado, o valor aplicado mês a meso, e o tempo a ser aplicado (em meses). Diante de todos esses dados, o programa deverá informar o valor total a ser resgatado no final da aplicação.

Exercício: Identifique os atributos e métodos desse cenário e crie uma struct para resolver o problema proposto acima.

Sugestão: crie um enumerador para os tipos de aplicação.

6. Programação orientada a objetos: classes e métodos

À medida que os sistemas crescem tornando-o cada vez mais complexos, torna-se mais difícil satisfazer a um número cada vez maior de requisitos desses sistemas. Atualmente, o paradigma orientado a objetos tem sido empregado no desenvolvimento de sistemas de software complexos e de grande porte. Neste sentido, a programação orientada a objetos (POO) tem papel fundamental.

Linguagens com Pascal, C, Basic, entre outras, são linguagens de programação procedimentais ou estruturadas. Isto é, cada declaração na linguagem informa que o computador deve realizar alguma tarefa como, por exemplo, ler um dado de entrada, adicionar uma valor a uma variável, realizar um operação artimética por algum número e exibir o resultado. Dessa forma um programa em uma linguagem estruturada como o C, nada mais é que uma lista de instruções executadas de forma seqüencial.

Para programas pequenos e de baixa complexidade, as linguagens acima dão conta do recado, afinal o programador cria uma lista de instruções e o computador as executa. Porém, à medida que os programas vão se tornando maiores, entender e tratar uma única lista de instruções fica mais difícil. Uma das soluções para resolver esses problemas seria quebrar o programa, ou seja, dividi-lo em partes menores que são chamadas de ***funções***. Por esta razão, a função foi adotada com uma forma de tornar os programas mais compreensíveis. Em outras linguagens, o mesmo conceito pode ser referido como uma subrotina, um subprograma ou um procedimento. Essa técnica de dividir o programa em partes menos é chamada de programação modular.

Acontece que, à medida que os programas se tornam maiores e mais complexos, até mesmo a programação modular começa a se mostrar limitada para resolver os problemas. Neste contexto, a POO apresenta-se como uma abordagem de programação que permite os programadores raciocinar e solucionar problemas em termos de objetos, os quais estão diretamente associados às entidades do mundo real. Como resultado desse mapeamento utilizando a POO, um programador desenvolve os objetos que compõem o sistema, ao invés de desenvolver o sistema como um conjunto de procedimentos e dados.

Sendo assim, a orientação a objetos, também conhecida como Programação Orientada a Objetos (POO) é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos. Um projeto orientado a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de software. O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. Além disso, conceitos adicionais como classes, encapsulamento, polimorfismo e herança fazem parte desse paradigma.

Estudaremos agora os principais conceitos relacionados a POO.

6.1. Classes e objetos: conceitos

Uma classe é considerada um tipo de dado, como os tipos básicos pré-definidos (inteiro, real, string, etc.) em todas as linguagens de programação. Sendo assim da mesma forma que podemos criar uma variável do tipo "int" também podemos criar uma variável do tipo da classe. Esta variável da classe é chamada de **OBJETO**.

Ao definirmos uma classe, estamos fornecendo a estrutura para o objeto. Toda vez que declaramos uma variável originada de uma classe dizemos que estamos instanciando um objeto. Quando definimos uma classe estamos criando um modelo para criarmos objetos e/ou outras classes(herança). Para exemplificar classes e objetos, vamos considerar a seguinte situação: No mundo atual existem empresas e em cada empresas existem funcionários, e cada funcionário possui uma particularidade, mas todos possuem a mesma estrutura e comportamento. A definição de um modelo que descreve essa estrutura e esses comportamentos é nomeado como uma Classe. Se pensarmos em carro na POO, funcionario seria uma classe que possui a estrutura com: nome, sexo, idade, estado_civil, salario,etc. Os

comportamentos que um funcionário pode ter são: registrarPonto(), gozarFerias(), receberSalario(), registrarOcorrências, etc.

Uma vez definida a classe do tipo ***Funcionario***, fica mais fácil criar objetos que utilizem desta mesma estrutura. Basta instanciar objetos do tipo ***Funcionario*** que teremos, carros diferentes, porém, com estrutura e comportamento iguais. Um objeto deverá possuir uma identidade, estado e comportamento.

1. Identidade: Todo objeto possui uma identidade. Por exemplo, um novo objeto:

Funcionario func = new Funcionario () - é um objeto com a identidade func;

2. Estado: O estado são as estruturas de um objeto. Por exemplo, o objeto ***func*** possui o atributo salario. Quando definimos um valor para esse atributo, nós definimos isso como:

func.salario = 2600,56;

3. Comportamento: Acima, nós também vimos os comportamentos de um objeto (receberSalario(), registrarOcorrências()). Quando nós chamamos esses métodos de um objeto, ele realiza a ação que lhe foi solicitada:

func.registrarPonto();

6.2. Classes e Objetos: Implementação no C#

Para criarmos um arquivo para codificar uma classe em um projeto é necessário adicionar um novo item ao projeto atual. Podemos fazer isso clicando com o botão direito sobre o projeto e em seguida clicar em **Add -> New Item**.

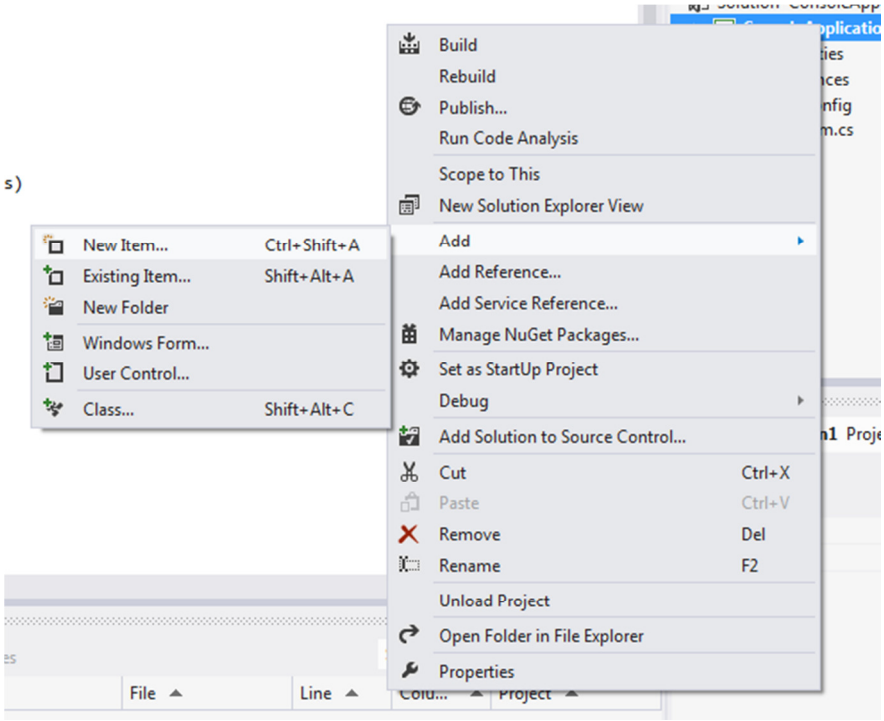


Figura 4 - Adicionando novo item ao projeto

Em seguida, selecione o item Classe e altere o nome para a sua classe. Veja na imagem abaixo:

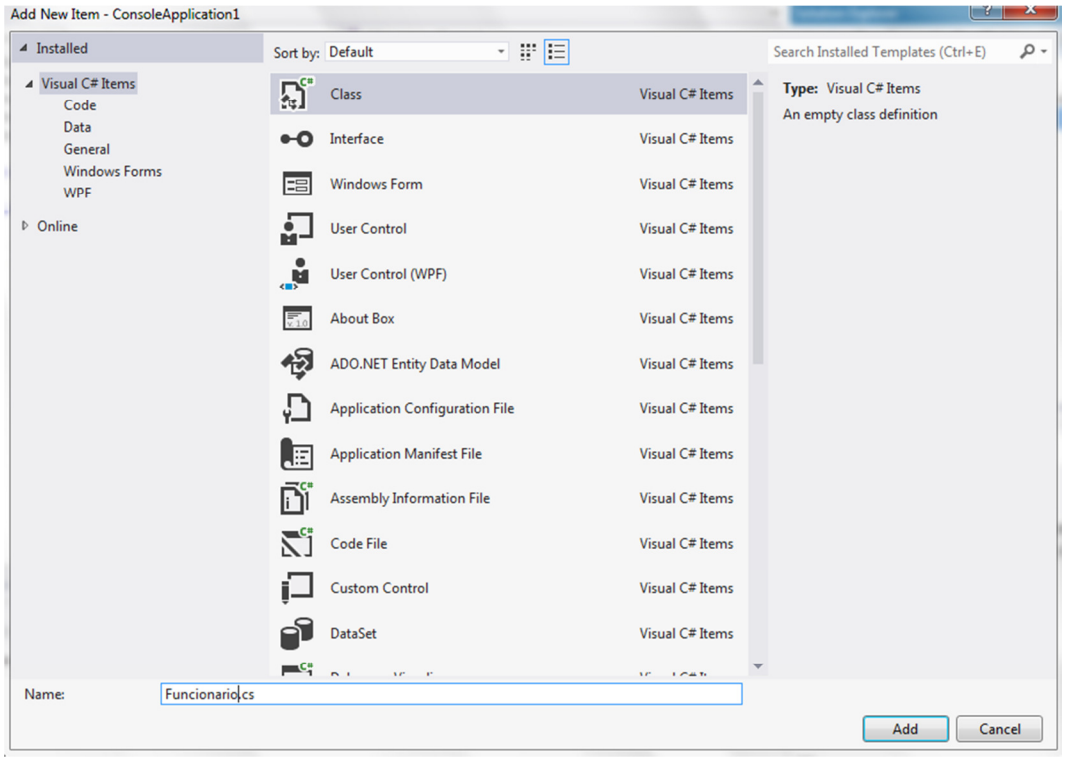


Figura 5 - Adicionando nova classe ao projeto

Depois de clicarmos em Add, será criado um novo arquivo. Observe que esse arquivo gerado já vem com algumas linhas de códigos com a estrutura da classe, mas para definir uma classe no C# utilizamos a palavra reservada ***class***. Uma classe é definida da seguinte forma:

```
class nomeDaClasse
  {
  }
```

Estudamos que objetos são instâncias de classe. Um programa orientado a objeto é composto por um conjunto de objetos que interagem entre si. Com a classe carro definida, podemos instanciá-la, ou seja, criar um objeto a partir da definição dessa classe. Veja o exemplo abaixo:

```
      Funcionario fun = new Funcionario ();
```

Uma classe é formada por métodos Construtores, Atributos, Propriedades e Métodos.

Inicialmente vamos falar sobre os Atributos. Os atributos são as características do objeto, como cor e tamanho, uma janela, por exemplo, tem atributos como o modelo, tamanho, abertura simples ou dupla, entre outros.

No caso do nosso exemplo Funcionario. Veja como ficariam os atributos da classe:

```
    class Funcionario
    {

        private string _nome;
        private int _idade;
        private string _sexo;
        private double _salario;

    }
```

Observe que estamos utilizando um modificar de acesso para os atributos. Nesse caso específico estamos utilizando o modificar `private`. Dessa forma, os atributos possuem visibilidade apenas na classe Funcionário. Se por um exemplo, instanciarmos um objeto Funcionario, não poderemos alterar os valores desses

atributos pois eles são privados. Podemos resolver esse problema de duas formas: A primeira seria coloca-los como atributos publicos utilizando nos atributos o modificador `public`, porém essa solução não é recomendada. A segunda forma seria declararmos propriedades para cada um dos atributos.

O foco importante em muitas linguagens de programação orientada a objetos é permitir controlar o acesso aos atributos internos de uma classe. Uma propriedade pode permitir o acesso somente à leitura e gravação de dados em atributos. Uma propriedade definida com ***get*** fornece acesso à leitura, enquanto uma propriedade definida como ***set*** fornece acesso à gravação. Você pode adicionar código para estas duas propriedades a fim de fornecer funcionalidade de leitura e gravação a esta propriedade.

Veja abaixo a classe Funcionario com seus atributos e propriedades:

```
class Funcionario
    {
        private string _nome;
        public string nome
        {
            get
            {
                return _nome;
            }
            set
            {
                _nome = value;
            }
        }

        private int _idade;

        public int idade
        {
            get
            {
                return _idade;
            }
            set
            {
                _idade = value;
            }
        }


    }
```

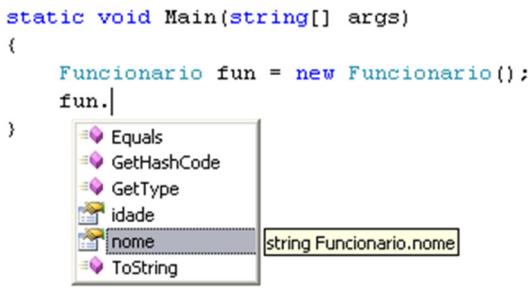
ATRIBUTO

PROPRIEDADES

ATRIBUTO

PROPRIEDADE

Ao instanciarmos uma classe, podemos acessar e atualizar os atributos das classes através dos métodos gets e set. Observe na imagem abaixo que depois de declarado o objeto fun, o intellisense automaticamente mostra as propriedades do objeto.



Podemos alterar o valor da propriedade nome, da seguinte forma

```
fun.nome = "Regilan Meira Silva";
```

Para exibir o valor atual da propriedade nome, podemos utilizar o Console.WriteLine para apresentar no prompt o valor da propriedade.

```
Console.WriteLine ("NOME DO FUNCIONARIO: " + fun.nome());
```

Se quisermos armazenar o valor da propriedade nome em outra variável podemos fazer da seguinte forma:

```
nomeDoFuncionario = fun.nome();
```

Veja agora o código completo:

```
static void Main(string[] args)
{
            string nomeDoFuncionario;
            Funcionario fun = new Funcionario();
            fun.nome = "Regilan Meira Silva";

            Console.WriteLine ("NOME DO FUNCIONARIO: " + fun.nome);
            nomeDoFuncionario = fun.nome;


}
```

Os métodos são semelhantes as funções, ou sejam os métodos servem para executar procedimentos, que podem ter ou não valores de retorno e parâmetros. Métodos que apenas executam um bloco de código sem retorno de valor recebem o

tipo “void” e métodos que retornam algum valor recebem o tipo de retorno. Ex: int, double, bool, etc. Ao utilizar métodos devemos também informar o modificar de acesso de método: private, public, protected, internal. (OBS: Na parte 6 desse tutorial, abordo sobre os modificadores de acesso.)

Vamos imaginar a seguinte situação para construir um método. Um funcionário possui um salário fixo mensal que está armazenado no atributo salario. Agora vamos construir dois métodos em nosso programa. Um método que calcula o décimo terceiro salário e um outro para calcular o salário das férias. Vejamos a codificação do primeiro método:

```
public double calculaDecimoTerceiro(int mesesTrabalhos)
{
        double decimoTerceiro;
        decimoTerceiro = _salario * mesesTrabalhos / 12;

        return decimoTerceiro;
}
```

No método acima chamado calculaDecimoTerceiro, é calculado o décimo terceiro salário de um funcionário, para isso é informado como parâmetro do método uma variável com o nome mesesTrabalhos referente aos meses trabalhados. Observe que dentro do método, utilizamos ***_salario (atributo)*** ao invés de ***salario (propriedade)***. Sempre que estivermos trabalhando dentro da classe, utilizamos o próprio atributo, quando estivermos usando um objeto instanciado a partir de uma classe, usamos a propriedade(get e set)

Agora veja a implementação do segundo método que é do tipo void.

```
public void calculaFerias()
{
        double ferias;
        ferias = _salario + _salario / 3;

        Console.WriteLine("O SALARIO DE FERIAS É R$: " + ferias);
}
```

O método acima calcula o salario de férias que é formado pelo salário do funcionário, acrescido de 1/3 do salario. No programa principal, devemos instanciar um objeto funcionario e chamar os métodos da classe. Veja como que fica:

```
Funcionario fun = new Funcionario();
```

```
  fun.nome = "Regilan Meira Silva";
  fun.salario = 2650;

  double dt;
  dt = fun.calculaDecimoTerceiro(5);

  Console.WriteLine("O funcionario: " + fun.nome + " receberá o
seguinte valor 13° salario R$: " + dt);

  fun.calculaFerias();
```

No exemplo acima, primeiramente foram alterados o nome e o salário de um funcionário. Em seguida foi criada uma variável com o nome ***dt*** para armazenar o décimo terceiro que é calculado pelo método ***calculaDecimoTerceiro***. Em seguida através do método ***WriteLine*** será exibido uma mensagem com o nome e o décimo terceiro salário do funcionário. Por último é chamado o método ***calculaFerias*** que apresenta o salário de férias do funcionário.

Uma classe possui um método especial que é chamado de Método Construtor. Métodos Construtores são métodos criados dentro das classes que são executados automaticamente no momento que instanciamos um objeto. No C# os métodos construtores devem ser métodos com o mesmo nome da classe. O método destrutor que destroi um objeto deve ser o nome da classe com um ~ (til) antes. Os construtores são definidos para no momento que instanciarmos um objeto, os valores dos seus atributos sejam alterados.

Veja como fica um construtor para a classe Funcionário:

```
public Funcionario(string nome,int idade,double salario,string sexo)
{
          _nome = nome;
          _idade = idade;
          _salario = salario;
          _sexo = sexo;
}
```

O construtor definido acima permite que quando o objeto for instanciado, seja informado os valores do nome,idade,salário e sexo de um determinado funcionário. Uma observação importante é que toda classe deve ter um construtor vazio, ou seja, um construtor para iniciar a classe sem alterar nenhum dos atributos. Por padrão se não definirmos um construtor o C# cria automaticamente um construtor vazio. Se criarmos um construtor, deveremos criar um construtor vazio para a classe.

```
//construtor vazio
public Funcionario()
{
}
```

Veja agora no programa principal que quando criarmos um objeto funcionario, poderemos iniciar a classe com os valores dos atributos:

```
Funcionario fun = new Funcionario();
2 of 2  Funcionario.Funcionario (string nome, int idade, double salario, string sexo)
```

```
Funcionario fun = new Funcionario("Regilan Meira Silva", 26, 3500,
"M");
```

Ao fazermos o código acima iniciamos a classe com os atributos alterados para os valores informados. Se em algum momento quisermos alterar o salario do funcionari, é só acessar a propriedade salário. Veja o exemplo abaixo:

```
Funcionario fun = new Funcionario("Regilan Meira Silva", 26, 3500,
"M");

double dt;
dt = fun.calculaDecimoTerceiro(5);

Console.WriteLine("O funcionario: " + fun.nome + " receberá o seguinte
valor 13° salario R$: " + dt);
fun.calculaFerias();

fun.salario = 6000;

dt = fun.calculaDecimoTerceiro(5);

Console.WriteLine("O funcionario: " + fun.nome + " receberá o seguinte
valor 13° salario R$: " + dt);

fun.calculaFerias();
```

No exemplo acima, primeiro foi instanciado uma classe ***Funcionario*** com os valores informado e logo após foram alterados o nome e o salário de um funcionário. Em seguida foi criada uma variável com o nome ***dt*** para armazenar o décimo terceiro que é calculado pelo método ***calculaDecimoTerceiro***. Em seguida através do método ***WriteLine*** será exibido uma mensagem com o nome e o décimo terceiro salário do funcionário. Por último é chamado o método **calculaFerias** que apresenta o salário de férias do funcionário. Por último é alterado o salario do

funcionario para R$ 6000 (fun.salario = 6000;) e novamente calculado o décimo terceiro e o salário de férias.

Com a classe pronta, podemos criar quantos objetos funcionário for necessários. Podemos ainda criar um vetor de objetos Funcionários. Agora imagine uma classe de funcionario pronta, poderemos utiliza-la para sempre em qualquer projeto que envolva a manipulação de dados do tipo funcionários/empregados/etc.

6.3. Herança

Herança pode ser definida como a capacidade de uma classe herdar atributos e comportamento de uma outra classe.

Um exemplo bem comum quando falamos de POO seria o objeto: CARRO. Carros de passeio, carros de corrida, carros conversíveis, carros com volante do lado esquerdo e outros do lado direito, carros de diferentes cores, fabricantes, etc. Cada vez que um novo carro é criado, não é necessário iniciar o projeto do ZERO, e começar todo um projeto novamente. Diversos conceitos abstraídos e encapsulados no objeto carro já foram planejados, testados e aprovados, e por isso não faz sentido gastar-se tempo e recursos para fazer tudo aquilo de novo. Todo carro terá um meio de propulsão, direção, acelerador e freio, rodas e pneus, e deste modo, todo novo carro pode herdar estas características e funcionamento de um projeto já existente. Caso seja necessário criar um novo carro que seja quase igual a todos os outros, mas que possua algumas coisas diferentes (por exemplo, utilizando um câmbio automático ao invés de manual), o projetista pode herdar o novo carro a partir do projeto básico, e então adicionar as características específicas do novo carro ao novo projeto.

O mesmo vale para outros objetos do dia-dia, como exemplo o objeto Pessoa em um cenário de uma escola. Uma pessoa poderia ser um estudante, um professor, um funcionário, um diretor, etc. Observe que nesse caso todos eles tem algumas características comuns: Nome, Documentação, Endereço, Data de Admissão, etc. Além das características eles também possuem comportamentos comuns: Registrar Ponto, Registrar Faltas, etc. Nesse caso poderiamos criar um projeto básico: CLASSE PESSOA e em seguida expandir a classe PESSOA com as características

particulares de cada tipo de pessoa: Professor (Área de Atuação, Formação, Disciplinas Ministradas, etc), Aluno (Curso, etc).

Sendo assim a herança é utilizada como o mecanismo para expressar a similaridade entre Classes, simplificando a definição de classes iguais que já foram definidas.

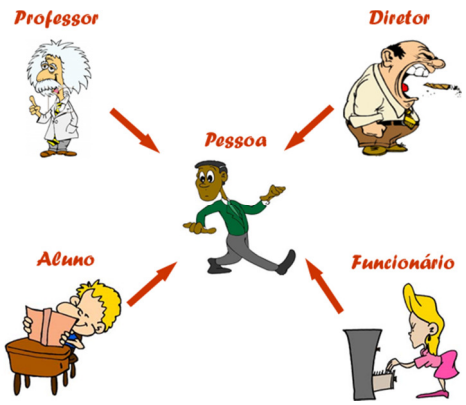


Figura 6 - Herança da classe pessoa

Herança é uma funcionalidade de C# que permite criar uma classe "mãe" que define características que todas as classes "filhas" também terão. Todas as propriedades, métodos e eventos da classe mãe serão herdadas pelas classes filhas, reaproveitando o código.

Para dizermos que uma classe herda de outra, utilizamos ":" (dois pontos) na definição da classe como descrito abaixo:

class <nomeDaClasseFilha> : <nomeDaClassePai>

Agora iremos começar a parte prática implementando uma classe base e em seguida criando novas classes a partir da classe base. Para isso, iremos utilizar o exemplo da classe PESSOA. Iniciaremos criando uma classe chamada **Pessoa**. Para a classe pessoa iremos definir os seguintes atributos: Nome, CPF e Sexo. Além disso, iremos incluir as propriedades gets e sets para cada atributo e criar um método comum a todas as classes derivadas que será o método **BaterPonto()**. Veja a seguir como ficaria a implementação da classe **Pessoa:**

```
class Pessoa
{
    //atributos
```

```
        private string _Nome;
        private string _CPF;
        private string _Sexo;

        //propriedades - gets e sets

        public string Nome
        {
            get
            {
                return _Nome;
            }
            set
            {
                _Nome = value;
            }
        }

        public string CPF
        {
            get
            {
                return _CPF;
            }
            set
            {
                _CPF = value;
            }
        }

        public string Sexo
        {
            get
            {
                return _Sexo;
            }
            set
            {
                _Sexo = value;
            }
        }

        public void BaterPonto()
        {
        }

    }
```

Depois de criado a classe base, iremos criar uma nova classe que será derivada da classe **Pessoa**. Essa classe será criada com o nome: **Professor**. Primeiro temos de declarar a nossa intenção de utilizar a classe **Pessoa** como base da classe **Professor**. Isso é realizado na classe derivada (Professor). A classe base é especificado, adicionando o caractere dois pontos (**:**), após a classe derivada e, em seguida, especificando a o nome da classe base. Na classe Professor iremos

adicionar dois atributos: ***AreaAtuacao*** e ***CargaHoraria***. Além disso incluiremos o método ***registrarDiario()***

```
    class Professor : Pessoa
    {
        private string _AreaAtuacao;
        private int _CargaHoraria;

        public string AreaAtuacao
        {
            get
            {
                return _AreaAtuacao;
            }
            set
            {
                _AreaAtuacao = value;
            }
        }

        public int CargaHoraria
        {
            get
            {
                return _CargaHoraria;
            }
            set
            {
                _CargaHoraria = value;
            }
        }

        public void registrarDiario()
        {
        }

      }
```

Agora iremos criar a classe **Aluno** que também é derivada da classe **Pessoa**. Sendo assim, a classe **Aluno** terá os mesmos atributos da classe **Pessoa**, além dos atributos específicos de um aluno: ***Serie e Turma***. Veja a implementação a seguir:

```
class Aluno : Pessoa
{
        private string _Serie;
        private string _Turma;

        public string Serie
        {
            get
            {
                return _Serie;
```

```
            }
            set
            {
                _Serie = value;
            }
        }

        public string Turma
        {
            get
            {
                return _Turma;
            }
            set
            {
                _Turma = value;
            }
        }
}
```

Com as classes implementadas, é hora de consumi-las no programa principal, no nosso caso um programa do tipo console. Veremos que o processo de criação dos objetos a partir das classes criadas (base e derivada) é o mesmo até agora. Veja a seguir alguns exemplos usando as classes criadas: ***Pessoa, Aluno e Professor***.

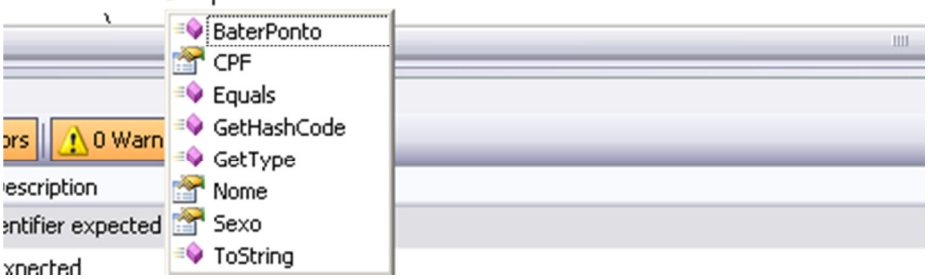
Iniciaremos criando um objeto para a classe ***Aluno***. Veja que no momento que criamos o objeto ***(a)*** para a classe ***Aluno***, todos os atributos e métodos criados na classe ***Pessoa*** também está disponível para a classe ***Aluno***.

```
class Program
{
    static void Main(string[] args)
    {
        Aluno a = new Aluno();

        a.
    }
}
```

BaterPonto
CPF
Equals
GetHashCode
GetType
Nome
Serie
Sexo
ToString
Turma

Em seguida iremos alterar os valores dos atributos do objeto Aluno. Por último criaremos mais um objeto, só que nesse caso, iremos criar um objeto para a classe Professor. Veja na imagem a seguir como que ficou o programa Principal.

```
class Program
{
    static void Main(string[] args)
    {
        Aluno a = new Aluno();

        a.Nome = "Aline Araujo Freitas";
        a.Sexo = "F";
        a.CPF = "000.000.000-00";
        a.Serie = "3° Ano";
        a.Turma = "A";

        Professor p = new Professor();

        p.Nome = "Regilan Meira Silva";
        p.Sexo = "M";
        p.CPF = "111.111.111-11";
        p.CargaHoraria = 40;
        p.AreaAtuacao = "Informática";

        p.BaterPonto();
        p.registrarDiario();

        Pessoa pe = new Pessoa();

        pe.
```



Até agora utilizamos uma classe base para nos permitir evitar repetição de código no mecanismo de herança. Utilizando a idéia de classe, sempre devemos ter alguma forma de implementação da classe base para que possamos reutilizar esse código nas classes derivadas. No C# existe um mecanismo de herança que permite criar as classes bases através de um tipo de contrato, esse tipo de contrato é conhecido como Interface.

Uma interface nada mais é do que um bloco de código que define um tipo e os métodos e propriedades que esse tipo deve possuir. Na prática o que acontece é que qualquer classe que quiser ser do tipo definido pela interface deve implementar os métodos dessa interface.

A interface não contém nenhum código de implementação, apenas assinaturas de métodos e/ou propriedades que devem ter seu código implementado nas classes que implementarem essa interface. Não irei trabalhar com interfaces nesse curso de C$, portanto para quem quiser entender um pouco mais sobre interfaces, sugiro fazer uma pesquisa sobre o assunto.

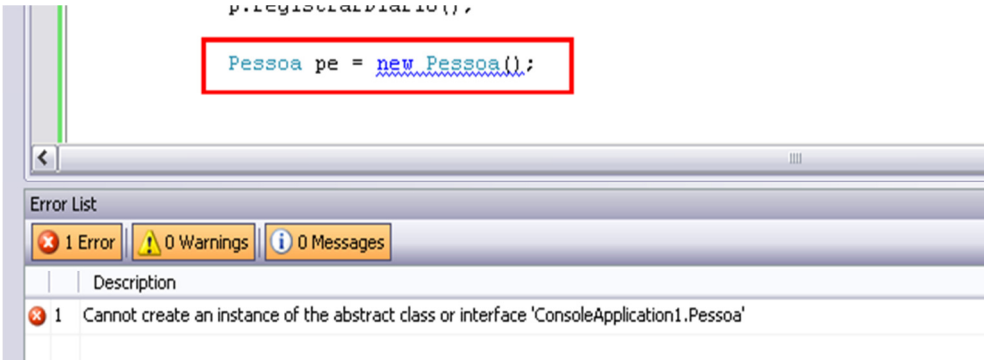
Outra forma bastante comum de implementação de Herança, é a criação de Classes abstratas. As classes abstratas devem ser usadas como classes bases (ou classes "mães") e não podem ser instanciadas, devendo então ser herdadas por alguma outra classe para podermos ter a implementação completa das mesmas.

Veja o exemplo de classe abstrata utilizando a classe Pessoa e Aluno.

Na classe pessoa iremos incluir a palavra reservada: ***abstract***. Dessa forma, a classe Pessoa é configurada como abstrata, não podendo ser instanciada. Veja a classe pessoa:

```
abstract class Pessoa
```

Em seguida, no programa principal quando tentamos iniciar um objeto do tipo Pessoa, o Visual C# detecta o erro, já que a classe Pessoa é do tipo abstrata.



6.4. Polimorfismo

Polimorfismo se refere à capacidade de criação de múltiplas classes com funcionalidades diferentes, mas que possuem os mesmos métodos e propriedades, só que implementados de maneiras diferentes. Polimorfismo é essencial para programação orientada a objetos pelo fato de permitir o uso de itens com mesmo nome, não importando o tipo do objeto que está em uso. Por exemplo, se tivermos uma classe base Pessoa e cada classe filha possuir o método **BaterPonto()**. Polimorfismo permite que tenhamos diferentes implementações do método BaterPonto () em cada classe filha mas que sejam utilizados por outros objetos ou métodos exatamente da mesma forma, independente de qual tipo o objeto é. Por exemplo, polimorfismo nos permite usar o método BaterPonto () da classe base Pessoa da mesma forma que chamamos o método BaterPonto da classe filha Professor.

Para exemplificar, utilizaremos as mesmas classes criadas até agora: Pessoa, Professor e Aluno. Para podermos sobrescrever um método, ele deve permitir a sobrescrita tendo sido declarado como "virtual" na sua declaração. O método que vai sobrescrever deve ter a palavra chave "override" na sua declaração. Veja na classe Pessoa como ficaria esse método:

```
    class Pessoa
    {
        public virtual void BaterPonto()
        {
            //Exemplo para ilustração do uso de polimorfismo
            Console.WriteLine("Método BaterPonto da classe Pessoa");

        }
```

Agora os métodos BaterPonto na classe Professor e Aluno:

```
class Professor : Pessoa
    {

        public override void BaterPonto()
        {
            //Exemplo para ilustração do uso de polimorfismo
            Console.WriteLine("Método BaterPonto da classe
Professor");
        }
```

```
class Aluno : Pessoa
    {

        public override void BaterPonto()
        {
            //Exemplo para ilustração do uso de polimorfismo
            Console.WriteLine("Método BaterPonto da classe Aluno");

        }
```

Depois de implementado o código acima, iremos testa-lo no programa principal. Abaixo temos a chamada dos métodos BaterPonto para cada Classe.

```
    class Program
    {
        static void Main(string[] args)
        {
            Aluno a = new Aluno();
            Professor p = new Professor();
            Pessoa pe = new Pessoa();

            a.BaterPonto();
            p.BaterPonto();
            pe.BaterPonto();

            System.Threading.Thread.Sleep(10000);

        }


      }
```

Agora executando o código acima:

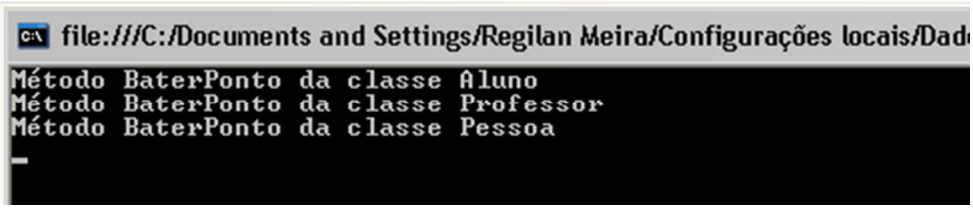


Figura 7 - Execução de código usando polimorfismo

6.5. Exercícios

Questão 01. Escreva uma classe chamada Quadrado com um atributo lado do tipo double. Essa estrutura deverá possuir dois métodos que devem retornar a área e o perímetro do quadrado segundo a fórmula abaixo:

- Area = lado²
- Perimetro = 4 * lado

Questão 02. As informações a seguir referem-se a uma conta de luz. Para cada conta de luz cadastra-se: data em que a leitura do relógio de luz foi realizada, número da leitura, quantidade de Kw gasto do mês, valor do Kw, data de vencimento, data do pagamento.

Nessa classe deverá também existir um método que calcula o valor a ser pago pelo cliente que é calculado da seguinte forma:

- Se o cliente pagar a conta em dias, o valor a ser pago é a quantidade de Kw gasto no mês multiplicado pelo valor do Kw.
- Se o cliente pagar a conta atrasado, o valor a ser pago é a quantidade de Kw gasto no mês multiplicado pelo valor do Kw, acrescentado um juros de 2% do valor da conta para cada dia em atraso.

Exercício: Identifique os atributos, métodos, propriedades e construtores desse cenário e crie uma classe para resolver o problema proposto acima. Depois da classe pronta, faça um programa para ler os dados e exibir na tela o valor a ser pago pelo cliente.

Questão 03. As informações a seguir se referem a uma aplicação de controle pessoal de horário de remédios, existente no IPhone de Regilan. Para cada remédio cadastra-se: o nome de quem vai tomar o remédio, a data de início, a quantidade de dias que foi prescrita pelo médico, a quantidade de vezes ao dia, a dosagem e o nome do remédio. Ao cadastrar o remédio, a aplicação sugere todos os horários possíveis para tomar o remédio. O usuário escolhe o melhor horário e a aplicação avisa até quando o remédio deve ser tomado e exibe na tela do IPhone os horários.

Exercício: Identifique os atributos, métodos, propriedades e construtores desse cenário e crie uma classe para resolver o problema proposto acima.

Questão 04. As informações abaixo se referem a algumas aplicações bancárias existentes no Banco ABC. O banco ABC é um banco de investimentos e possui algumas aplicações onde cada uma possui uma taxa de juros diferente, conforme a tabela abaixo.

Obs.: a/m significa ao mês.

- Poupança (Pode-se resgatar o valor aplicado a qualquer momento) – 0,78 a/m
- Titulo de Capitalização (Pode-se resgatar após 24 meses) – 0,97 a/m
- CDB (Pode-se resgatar após 6 meses) – 0,85 a/m
- RDB (Pode-se resgatar a metade aplicada em seis meses e após 12 meses poderá ser resgatado a qualquer momento) – 0,90 a/m

O cliente deverá escolher uma aplicação, o valor inicial aplicado, o valor aplicado mês a meso, e o tempo a ser aplicado (em meses). Diante de todos esses dados, o programa deverá informar o valor total a ser resgatado no final da aplicação.

Exercício: Identifique os atributos, métodos, propriedades e construtores desse cenário e crie uma classe para resolver o problema proposto acima.

Sugestão: crie um enumerador para os tipos de aplicação.

Questão 05. Seguindo a especificação abaixo, desenvolva em C# três classes: Pessoa, PessoaFisica e PessoaJuridica.

- Pessoa: Classe abstrata e deverá ter um método virtual de nome CalcularTarifa que deverá retornar um double e não receber parâmetros.
- PessoaFisica: deverá herdar de Pessoa, ter duas propriedades públicas Nome e Saldo respectivamente do tipo string e double além de implementar o método CalcularTarifa que deverá retornar Saldo multiplicado por 0.1.
- PessoaJuridica: deverá herdar de Pessoa, ter também duas propriedades públicas RazaoSocial (string) e Limite (double não negativo). A

implementação do método CalcularTarifa deverá retornar Limite multiplicado por 0.2.

Questão 06. Construir uma classe para representar um produto com os seguintes dados: nome e preço. Além dos construtores e propriedades de obter e passar dados (gets e sets), faça um método que efetua a leitura dos dados (atributos) e um método para calcular o imposto. O imposto do produto deve ser calculado da seguinte maneira: se o preço for menor que R$ 100,00 o imposto é de 5%, se for maior que R$ 100,00 o imposto é de 10%.

Depois de implementado a classe Produto, faça uma classe para representar um produto importado. A classe Produto Importado herda da classe produto, e possui os seguintes atributos: PaisOrigem, TarifaImportação.

Questão 07. Faça uma classe para representar um funcionário com os seguintes dados nome, horas trabalhadas e valor pago por hora trabalhada. Além do construtor e propriedades de obter e passar dados (gets e sets), faça um método para calcular o salário final e um método para ler os dados (atributos).

A seguir faça uma classe para representar um funcionário sênior. A diferença entre um funcionário sênior e um funcionário normal é que a cada 10 horas trabalhadas o funcionário sênior recebe um bônus de 10% do valor pago por hora trabalhada. Faça essa classe por meio de herança e reimplemente o método calculaSalario

7. Interface gráfica com usuário (GUI)

Uma GUI (Graphical User Interface) permite que os usuários interajam com um programa visualmente. O Visual C# fornece aos diferentes aplicativos um conjunto consistente de componentes visuais de interface com o usuário.

Os aplicativos para Windows Desktop permitem criar interfaces gráficas para os programas. Um formulário é um elemento gráfico que aparece na área de trabalho. Um formulário pode ser uma caixa de diálogo, uma janela com diversos controles (botões, caixa de texto, rótulos, listas, etc.).

No Visual C# Express, os controles gráficos estão contidos na Toolbox (caixa de ferramentas). Quando o usuário adiciona um controle em um formulário, ele poderá trabalhar com as propriedades e os eventos que esse controle possui.

Para desenvolver um programa com interface gráfica no Visual C#, é necessário escolher um projeto do tipo WINDOWS FORMS APPLICATION no momento que for criado um novo projeto. Veja a imagem abaixo:

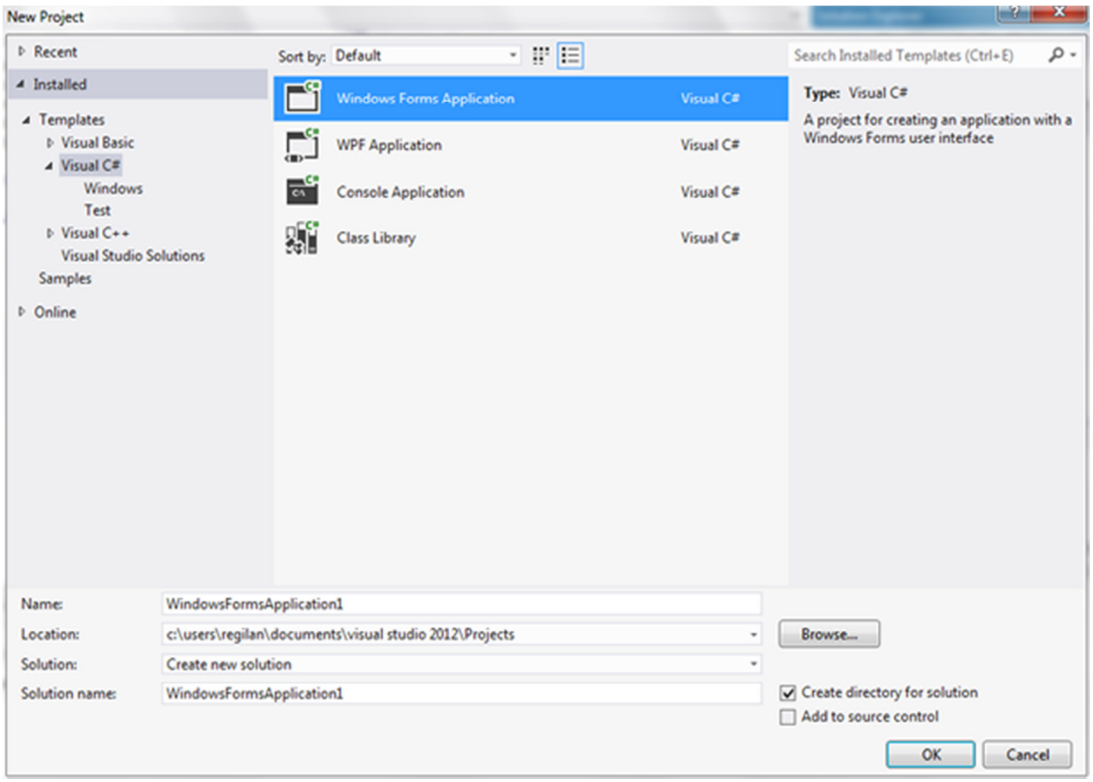


Figura 8 - Criando um Windows Forms Application

Em seguida, após clicar o botão OK, o Visual C# automaticamente criará um novo projeto com um formulário padrão (OBS: Um projeto normalmente é formado por mais de um formulário)

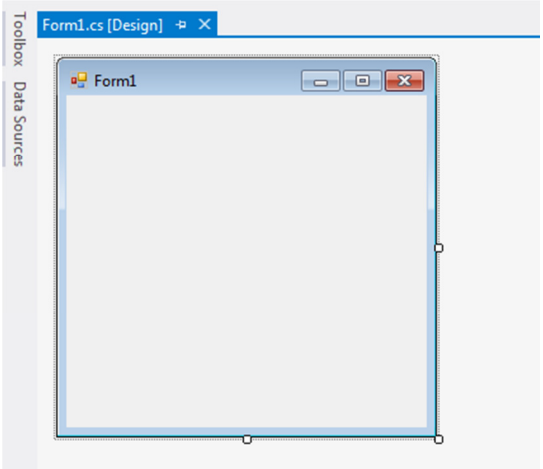


Figura 9 - Formulário padrão em aplicações Windows

Um controle visual é caracterizado por possuir Propriedades e Eventos. As propriedades identificam as características de um controle: cor, tamanho, texto, cor da fonte, cor de fundo, etc. Os eventos identificam o comportamento/ação que um controle possui: ao clicar o mouse, ao teclar a tecla ENTER, ao arrastar o mouse, etc.

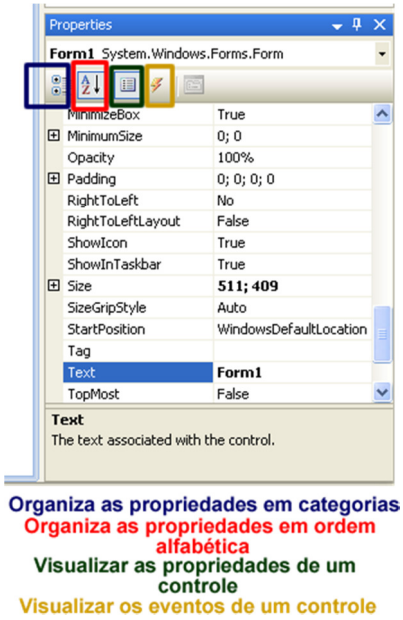


Figura 10 - Janela de propriedades

Existe uma propriedade importante que é presente em todos os controles visuais, que é a propriedade **name**. Essa propriedade é utilizada para identificar (dar um nome) a um controle. Durante a programação é comum utilizarmos um controle dentro de um bloco de código. O valor que é informado na propriedade **name** é que será utilizado para referenciar o controle no código fonte.

O Controle Formulário pode ser considerado o principal componente visual de uma aplicação do tipo Windows Forms Application, visto que ele funciona como uma espécie de container dos demais controles, ou seja, para que outros controles visuais sejam utilizados, é necessário inserir os demais componentes visuais em um formulário.

O controle Formulário possui uma série de eventos e propriedades.

Propriedades:

- Name: nome do controle. O valor informado na propriedade name será usado no código fonte para referenciar um determinado componente
- AcceptButton: configura um botão presente no formulário para que ao ser pressionado a tecla ENTER, o código presente no evento CLICK do botão seja acionado
- BackColor: configurar uma cor de fundo para o formulário
- BackgroundImage: configura uma imagem como fundo de um formulário
- CancelButton: configura um botão presente no formulário para que ao ser pressionado a tecla ESC, o código presente no evento CLICK do botão seja acionado.
- ControlBox: retira os botões padrões de um formulário (maximizar,minimizar,fechar)
- Cursor: define o cursor do mouse que estará visível quando o mouse estiver sobre o formulário
- Icon: define um ícone para o formulário
- IsMDIContainer: configura um formulário como um container de formulário.
- MaximizeBox: desabilita o botão maximizar do formulário
- MinimizeBox: desabilita o botão minimizar do formulário
- Size: define o tamanho do formulário

- StartPosition: define a posição inicial do formulário (Exemplo: centralizado)
- Text: define um texto para a barra de título do formulário
- WindowState: define o estado inicial da janela (maximizado,minimizado,etc)

Eventos Principais

- Click: evento disparado quando o usuário clicar no formulário
- FormClosed: evento disparado quando o usuário fechar o formulário
- Load: evento disparado quando o usuário carregar o formulário

Para adicionar um componente visual ao formulário, é necessário escolher o componente a ser inserido através da ToolBox. A ToolBox é organizada através de categorias, sendo a mais importante a categoria Common Controls que agrupa os controles mais utilizados. Para inserir um controle no formulário, clique no controle desejado e arraste-o até o formulário, ou dê um clique duplo no controle.

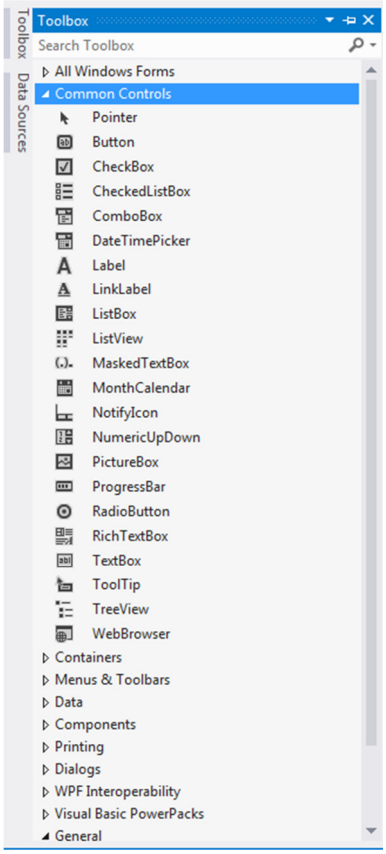


Figura 11 – Toolbox

Os controles disponíveis na aba Common Controls refere-se aos controles mais populares presentes em aplicações do tipo Windows. Exemplos: botões, caixa de texto, lista de seleção, entre outros.

7.1. Controle Label, TextBox, Button

O controle label (rótulo) fornece instruções textuais ou informações sobre o programa. Um label apresenta texto somente de leitura ou texto que o usuário não pode modificar.

O controle textbox(caixa de texto) é uma área em que o texto pode ser fornecido pelo usuário, a partir do teclado.

O controle button (botão) é um controle em que o usuário clica para ativar uma ação específica. Um programa poderá usar vários botões.

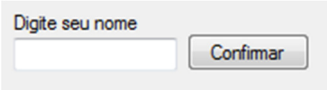


Figura 12 - Controles label, textbox e button

Principais propriedades da label:

- Font: a fonte usada pelo texto no label
- Forecolor: a cor do texto usada no label
- Image: configura uma imagem de fundo para a label
- Text: o texto que vai aparecer no label
- TextAlign: o alinhamento do texto do label no controle

Principais eventos da label:

- Click: quando o usuário clicar sobre o controle
- MouseMove: quando o usuário mover o cursor do mouse sobre o controle

Principais propriedades da TextBox:

- BackColor: cor de fundo da textbox
- BorderStyle: definr o estilo da borda da caixa de texto
- BeepOnError: emite um alerta sonoro caso o usuário digitar uma letra ao invés de número
- Cursor: define o cursor do mouse que será utilizado quando o mouse estiver sobre o controle
- Enabled: habilita/desabilita um controle
- Font: a fonte usada pelo texto no textbox
- Forecolor: a cor do texto usada no textobox
- Mask: define o formato da máscara. Utilize a # para os dados que serão informados pelo usuário. Exemplo: Caso seja feita uma máscara para data, a propriedade mask deveria ficar assim: ##/##/####
- PasswordChar: define um caractere para substituir o caractere digitado
- ScrollBars: define uma barra de rolagem para o controle
- Size: define o tamanho do controle
- Text: o texto que vai aparecer no label
- TextAlign: o alinhamento do texto do label no controle
- Visible: configura se o controle está visível ou não no formulário

Principais eventos da TextBox:

- Click: quando o usuário clicar sobre o controle
- DoubleClick: quando o usuário dar um click duplo do mouse sobre o controle
- KeyPress: quando uma tecla for pressionada sobre o controle
- MouseMove: quando o usuário mover o cursor do mouse sobre o controle
- TextChanged: quando for alterado o texto que estiver no controle

Principais propriedades de um botão:

- BackColor: cor de fundo do botão
- BackgroundImage: define uma imagem como fundo do botão
- Cursor: define o cursor do mouse que será utilizado quando o mouse estiver sobre o controle

- Enabled: habilita/desabilita um controle
- FlatStyle: estilo do botão
- Font: a fonte usada pelo texto no botão
- Forecolor: a cor do texto usada no botão
- Image: adiciona uma imagem a um botão
- ImageAlign: alinhamento da imagem no botão
- Size: define o tamanho do controle
- Text: o texto que vai aparecer no botão
- TextAlign: o alinhamento do texto do botão
- Visible: configura se o controle está visível ou não no formulário

Principais eventos do botão:

- Click: quando o usuário clicar sobre o controle

Um outro componente, usado para inserir imagens em um formulário é o Picturebox. Uma caixa de figura (PictureBox) tem a função de exibir uma imagem. A imagem, configurada através da Propriedade Image pode estar em diversos formatos: .bmp,gif,jpg, etc.

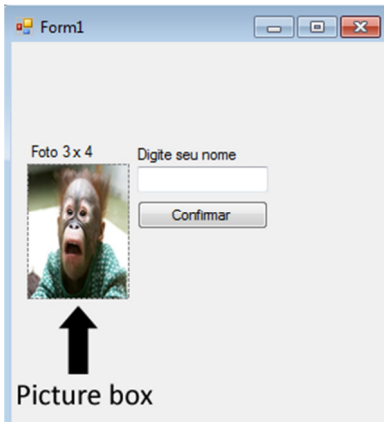


Figura 13 - Picture box

Principais propriedades da PictureBox:

- Image: imagem a ser exibida
- SizeMode:controla o dimensionamento e o posicionamento da imagem
- Size: tamanho da imagem

- Visible: visibilidade do controle

Principais eventos da PictureBox

- Click: disparado quando o usuário clica no controle

As MessageBox, ou caixa de diálogos, são janelas que exibem mensagens importantes para o usuário de um aplicativo.

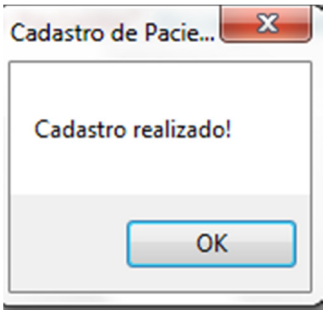


Figura 14 - Exemplo de messagebox

Para exibir uma mensagem através de uma MessageBox, utilize o comando: MessageBox.Show (mensagem,título,botões,ícones).

A caixa de diálogo possui 4 argumentos:

- Mensagem: mensagem a ser exibida
- Título: texto apresentado na barra de títulos
- Botão: valor que indica quais botões devem aparecer
- Ícone: ícone a ser exibido na mensagem

7.2. ComboBox, listBox, checkbox, radiobutton, maskedtextbox e controles para datas

O controle ListBox(caixa de lista) permite que o usuário veja e selecione dentre múltiplos itens em uma lista.

Caso o número de itens for grande demais para ser exibido simultaneamente, uma barra de rolagem aparecerá automaticamente no controle.

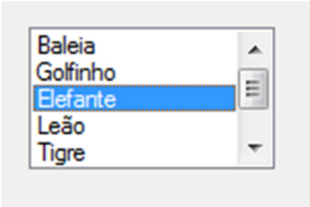


Figura 15 – Listbox

Principais propriedades da ListBox:

- Items: relaciona a coleção de itens dentro da listbox
- MultiColumn: indica se a ListBox pode ser dividida em múltiplas colunas. As múltiplas colunas são usadas para tornar as barras de rolagem verticais desnecessárias
- SelectedIndex: retorna o índice do item atualmente selecionado
- SelectedIndices: retorna uma coleção dos itens atualmente selecionados
- SelectedItem: retorna uma referência para o item selecionado. Se múltiplos itens forem selecionados, ela retornará o item com o menor número de índice
- SelectedItems: Retorna uma coleção dos items atualmente selecionados
- SelectionMode: determina o número de itens que podem ser selecionados
- Sorted: Indica se os itens aparecem em ordem alfabética

Principais eventos da listbox:

- Click: disparado quando o usuário clicar sobre o controle
- SelectedIndexChanged: disparado quando o índice selecionado muda.

O controle combobox (caixa de combinação) combina os recursos de TextBox com uma lista suspensa. A lista suspensa contém uma lista a partir da qual valores podem ser escolhidos. Normalmente ela aparece como uma caixa de texto com uma seta para baixo a sua direita

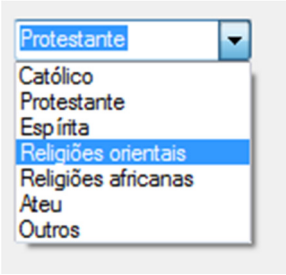


Figura 16 – Combobox

Principais propriedades da ComboBox:

- DropDownStyle: determina o tipo de caixa de combinação. Os tipos podem ser: Simple,DropDown e DropDownLista
- Items: coleção de itens no controle ComboBox
- MaxDropDownItens: especifica o número máximo de itens que podem aparecer na lista suspensa.
- SelectedIndex: retorna o índice do item atualmente selecionado
- SelectedItem: retorna uma referência para o item atualmente selecionado

Os eventos da ComboBox são semelhantes aos da ListBox.

Principais eventos da ComboBox :

- Click: disparado quando o usuário clicar sobre o controle
- SelectedIndexChanged: disparado quando o índice selecionado muda

O Visual C# tem dois tipos de botões de estado:

- CheckBox: caixa de seleção
- RadioButton: botão de opção

Os botões de estado podem estar no estado ligado/desligado ou verdadeiro/falso. Os dois tipos de botões tem a mesma função, mas funcionam de forma diferente.

O checkbox ou caixa de seleção é um pequeno quadrado branco que pode estar vazio ou conter uma marca de visto. Quando uma caixa de seleção está selecionada, uma marca de visto preta aparece nela. Não existem restrições sobre como as caixas de seleção são utilizadas: qualquer número pode ser selecionado em dado momento, O texto que aparece ao lado de uma caixa de seleção é denominado rótulo da caixa de seleção.

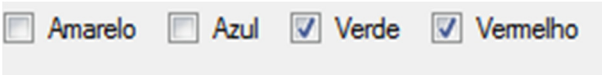


Figura 17 – Checkbox

Principais propriedades da CheckBox:

- Checked: Indica se a CheckBox foi marcada
- CheckedAlign: Indica o local da caixa de seleção dentro do controle
- CheckedState: Indica o status da CheckBox que pode ser: Checked, Unchecked ou Indeterminate
- Text: texto exibido à direita da CheckBox(rótulo)

Principais eventos da CheckBox:

- CheckedChanged: disparado sempre que a checkbox é marcada ou desmarcada.
- CheckedStateChanged: disparado quando a propriedade CheckState muda

Os botões de opção (radiobutton) são semelhantes as caixas de seleção, pois eles também tem dois estados: selecionado e não selecionado. Entretanto, os botões de rádio normalmente aparecem como um grupo no qual apenas um botão pode ser selecionado de cada vez. Selecionar um botão de opção diferente no grupo obriga todos os outros botões de rádio do grupo a perderem a seleção.

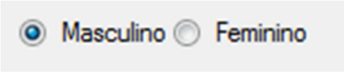


Figura 18 – Radiobutton

Principais propriedades do RadioButton:

- Checked: Indica se o radiobutton foi marcado
- CheckedAlign: Indica o local da caixa de seleção dentro do controle
- Text: texto exibido à direita do RadioButton (rótulo)

Principais eventos do RadioButton :

- CheckedChanged: disparado sempre que o radiobutton é marcado ou desmarcado.
- Click: disparado quando o usuário clica no controle

O controle CheckedListBox (caixa de lista marcada) deriva da classe ListBox e inclui uma caixa de seleção ao lado de cada item da lista. Os itens poderão ser adicionados por meio da smart tag: EditItems ou através da propriedade Items.

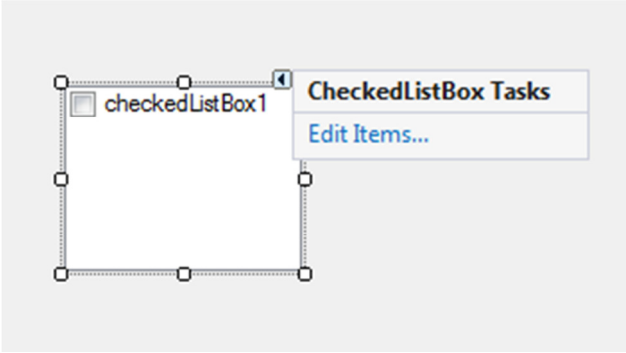


Figura 19 – Checklistbox

Principais propriedades do CheckListBox:

- CheckedItems: relaciona a coleção de itens que estão marcados
- Items: adiciona e edita os itens da lista

- SelectionMode: determina quantos itens podem ser marcados. Os únicos valores possíveis são One (permite que marcações múltiplas sejam feitas) ou None (não permite que nenhuma marcação seja feita)

Principais eventos do CheckListBox

- Click: disparado quando o usuário clica no controle
- ItemCheck: gerado quando um item é marcado ou desmarcado

O Controle MaskedTextBox tem funcionamento semelhante a uma caixa de texto, porém pode-se definir uma máscara de entrada de dados.

Exemplo: Um CPF tem o seguinte formato:

nnn.nnn.nnn-nn

n significa um número, dessa forma, quando o usuário digitar o CPF só será aceito números e os números digitados estarão no formado de CPF. Exemplo: 001.453.872-92

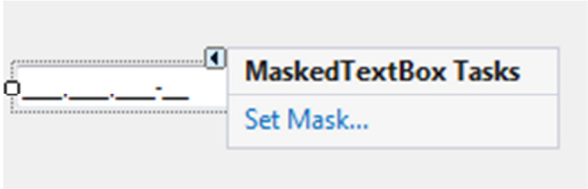


Figura 20 – Maskedtextbox

Principais propriedades do MaskedTextBox:

- BackColor: cor de fundo da textbox
- BorderStyle: definr o estilo da borda da caixa de texto
- BeepOnError: emite um alerta sonoro caso o usuário digitar uma letra ao invés de número
- Cursor: define o cursor do mouse que será utilizado quando o mouse estiver sobre o controle
- Enabled: habilita/desabilita um controle

- Font: a fonte usada pelo texto no maskedtextbox
- Forecolor: a cor do texto usada no maskedtextbox
- Mask: define o formato da máscara. Utilize a # para os dados que serão informados pelo usuário. Exemplo: Caso seja feita uma máscara para data, a propriedade mask deveria ficar assim: ##/##/####
- PasswordChar: define um caractere para substituir o caractere digitado

Os eventos do MaskedTextBox são semelhantes aos eventos do TextBox:

- Click: quando o usuário clicar sobre o controle
- DoubleClick: quando o usuário dar um click duplo do mouse sobre o controle
- KeyPress: quando uma tecla for pressionada sobre o controle
- MouseMove: quando o usuário mover o cursor do mouse sobre o controle
- TextChanged: quando for alterado o texto que estiver no controle

O controle MonthCalendar cria uma calendário mensal no formulário**. As principais propriedades são:**

- MinDate: define a menor data que poderá ser selecionada
- MaxDate: define a maior data que poderá ser selecionada
- TodayDate: data do dia

Os principais eventos são:

- DateChanged: disparado quando a data for alterada

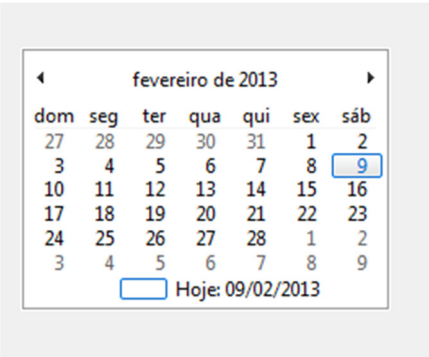


Figura 21 – Monthcalendar

O controle DataTimePicker é utilizado para o usuário escolher uma data a partir de uma lista suspensa, na qual exibe um calendário quando clicado.

As principais propriedades do DateTimePicker são:

- Format: define o formato da lista suspensa que poderá ser Long(Data Completa), Short (Data Curta), entre outros
- MinDate: define a menor data que poderá ser selecionada
- MaxDate: define a maior data que poderá ser selecionada
- Value: data e hora atual

Os principais eventos são:

- ValueChanged: disparado quando a data for alterada

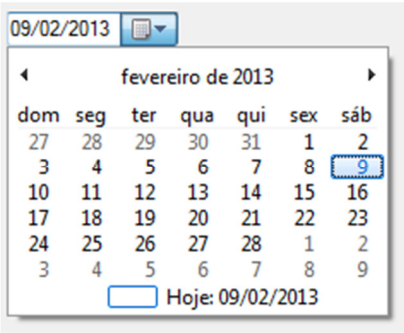


Figura 22 - Datetimepicker

O controle NumeriUpDown cria uma lista de números, onde através de botões o usuário poderá ir para o próximo número da lista ou para o número anterior da lista.

Figura 23 - Numericupdown

Principais propriedades:

- Increment: define o incremento da lista
- Minimun: define o menor valor da lista
- Maximun: define o maior valor da lista
- TextAlign: define o alinhamento do texto(número) na lista
- Value: retorna o valor atual

Principais eventos:

- Click: disparado quando o usuário clicar com o mouse sobre o controle
- ValueChanged: disparado quando o usuário alterar o valor da lista

7.3. GroupBox, panel, tabcontrol, listview, treeview, e outros

As groupbox(caixa de grupo) e panel(painéis) organizam componentes em uma interface gráfica. Por exemplo, os botões relacionados a uma tarefa em particular podem ser colocados dentro de uma GroupBox ou de um Panel.

A diferença entre os dois componentes é que as GroupBox podem exibir uma legenda e os Panels podem ter barras de rolagem.

Figura 24 – Groupbox

Figura 25 - Panel

Para criar uma groupbox ou um panel arreste-os até um formulário e coloque os demais controles dentro da groupbox ou do panel.

Esses controles são usados para organizar um formulário, dessa forma apesar de existirem eventos associados a estes componentes eles normalmente não são usados.

Principais propriedades do groupbox:

- Controls: os controles que a groupbox contém
- Backcolor: cor de fundo do groupbox
- Cursor: define o cursor do mouse que será utilizado quando o mouse estiver sobre o controle
- Enabled: habilita/desabilita um controle
- Text: texto exibido na parte superior da groupbox (legenda)
- Visible: configura se o controle estará visível ou não no formulário

Observação: quando a propriedade visível de um groupbox ou panel estiver com valor false, todos os controles que pertencem ao groupbox também estarão invisíveis no formulário.

Principais propriedades do Panel:

- AutoScroll: indica se as barras de rolagem aparecem quando o Panel é pequeno demais para os controles contidos nele. O padrão é false
- Backcolor: cor de fundo do painel
- BorderStyle: a borda do painel
- Controls: os controles que o panel contém
- Cursor: define o cursor do mouse que será utilizado quando o mouse estiver sobre o controle
- Enabled: habilita/desabilita um controle
- Visible: configura se o controle estará visível ou não no formulário

O controle TabControl (controle com guias) cria janelas com guias, como o exemplo abaixo:

Figura 26 – Tabcontrol

O controle TabControl permite ao programador projetar interfaces com o usuário que encaixem um grande número de controles ou um grande volume de dados, sem usar espaço valioso da tela

Os tabcontrols contêm objetos TabPage que permitem criar as guias do controle. Os programadores poderão adicionar tabcontrol clicando na smartag do controle conforme abaixo:

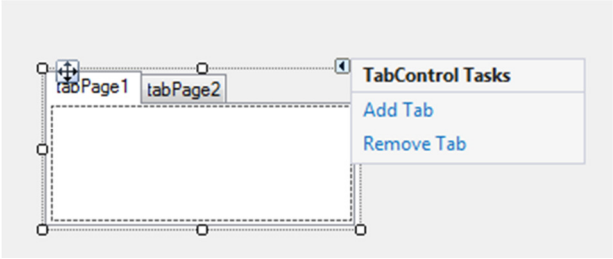


Figura 27 - Tabcontrol: adicionando guias

O programador também poderá adicionar as tabpages através da propriedade TabPage que define a coleção de guias que o controle pode ter.

As principais propriedades do TabControl são:

- Appearance: define a aparência visual do controle
- Multiline: indica se múltiplas linhas de guias podem ser exibidas
- SelectedIndex: indica o índice da tabpage que está selecionada
- SelectedTab: indica a tabpage que está selecionada
- TabPages: coleção de tabpages dentro da TabControl

O principal evento do controle é o SelectedIndexChanged que é gerado quando o SelectedIndex muda, ou seja, quando outra tabpage é selecionada

O controle TreeView mostra nós hierarquicamente em uma árvore. Os nós são objetos que contêm valores e podem se referir a outros nós. Um nó pode conter nós filhos, que podem ser progenitores de outros nós. Uma árvore é uma coleção de nós, normalmente organizados de maneira hierárquica. O primeiro nó progenitor de uma árvore é o nó raiz. Os controles TreeView são úteis para exibir informações hierárquicas, como, a estrutura de arquivos do Windows Explorer.

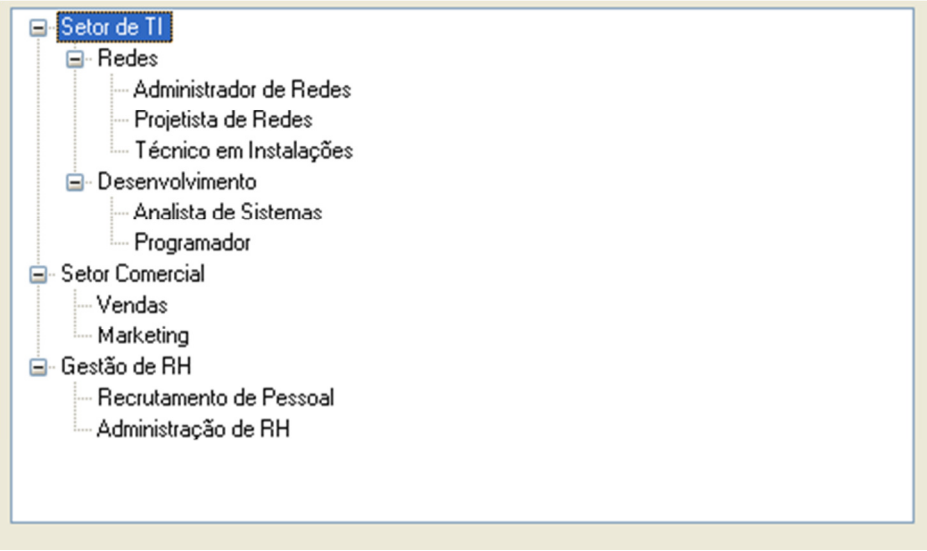


Figura 28 – Treeview

Um nó progenitor pode ser expandido ou recolhido clicando-se na caixa de sinal de mais ou menos a sua esquerda. Os nós sem filhos não possuem a caixa de expansão ou recolhimento. Os nós mostrados em uma TreeView são chamados de Nodes e são adicionados através da smartag (EditNode) da TreeView.

Figura 29 - Treeview: adicionando nós

As principais propriedades da TreeView são:

- Checkboxes: indica se as caixas de seleção aparecem ao lado dos nós
- ImageList: indica uma imagelist usada para exibir ícones ao lado dos nós. Uma imagelist é uma coleção que contém vários objetos do tipo image
- Nodes: relaciona a coleção de TreeNode no controle. Contém os métodos Add(adiciona um objeto TreeNode), Clear(exclui a coleção inteira) e Remove(exclui um nó específico)
- SelectedNode: o nó atualmente selecionado

O principal evento da TreeView é:

- AfterSelect: gerado após o nó selecionado mudar.

Os nós (TreeNode) fazem parte de uma TreeView, dessa forma, um treenode adicionado também possuem propriedades e eventos.

As principais propriedades do TreeNode são:

- FullPath: caminho do nó, começando na raiz da árvore
- ImageIndex: especifica o índice da imagem a ser exibida quando o nó perder a seleção
- LastNode: especifica o último nó da coleção Node
- NextNode: próximo nó irmão
- Nodes: a coleção de treenodes contidos no nó atual. Contém os métodos: Add,Clear,Remove.
- PrevNode: indica o nó anterior
- SelectedImageIndex: índice da imagem a ser usada quando o nó for selecionado

- Text: especifica o texto a ser exibido na TreeView

Os principais eventos do TreeNode são:

- Collapse: recolhe um nó
- Expand: expande um nó
- ExpandAll: expande todos os filhos de um nó
- GetNodeCount: retorna o número de nós filhos

Para adicionar nós via programação, devemos primeiro criar um nó raiz. Em seguida use o método Add para adicionar esse novo nó (treenode) na coleção de Nodes da TreeView. Dessa forma para adicionar um nó raiz faremos da seguinte forma:

```
nomeDaTreeView.Nodes.Add(new TreeNode(Texto ou Rótulo)
```

Para adicionar filhos em um nó raiz, adicione novos TreeNodes em sua coleção Nodes. Para selecionar o nó raiz apropriado da treeview, escrevemos:

```
nomeDaTreeView.Nodes[indice].Nodes.Add(new
TreeNode(Texto/Rótulo))
```

O controle ListView (visualização de lista) é semelhante a uma ListBox no sentido de que ambos exibem listas a partir das quais o usuário pode selecionar um ou mais itens. A diferença importante entre os dois controles é que a listview pode exibir ícones ao lado dos itens da lista de várias maneiras.

Veja um exemplo da utilização do ListView no Windows 7.

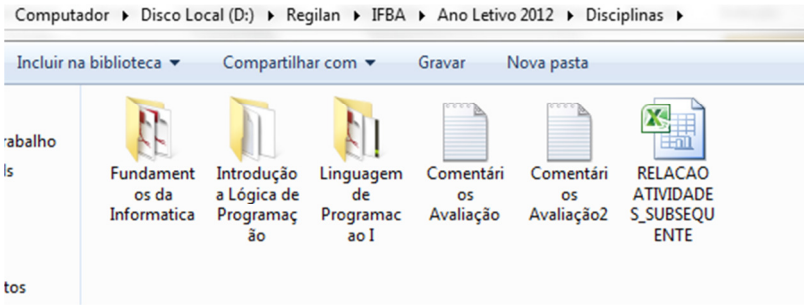


Figura 30 – Listview

As principais propriedades da ListView são:

- Activation: determina como o usuário ativa um item.
- CheckBoxes: indica se os itens aparecem com caixas de seleção
- LargeImageList: indica a imagelist usada ao exibir ícones grandes
- Items: retorna a coleção de itens atualmente selecionada
- Multiselect: determina se a seleção múltipla é permitida
- SelectedItems: relaciona a coleção de itens atualmetne selecionada
- SmallImageList: especifica a imagelist usada ao exibir ícones pequenos
- View: determina a aparência da ListView

O principal evento do ListView é o ItemActive que é gerado quando um item na ListView é ativado.

O controle NotifyIcon é usado para criar um ícone e uma mensagem de notificação na bandeja (barra de tarefas) do Windows. Na imagem abaixo foi usado um ícone de um símbolo de nota musical com uma mensagem. Assim que o programa estiver sendo carregado, o ícone aparecerá. Quando o curso do mouse estiver em cima do ícone, aparecerá um texto informativo.

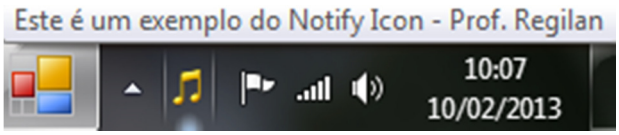


Figura 31 - Notify icon

As principais propriedades do NotifyIcon são:

- Icon: define o ícone a ser exibido
- Text: define o texto a ser exibido

O principal evento é o MouseDoubleClick que é gerado quando o usuário clicar duas vez com o mouse sobre o ícone.

O controle ProgressBar mostra uma barra de progresso que indica o progresso de uma determinada operação.

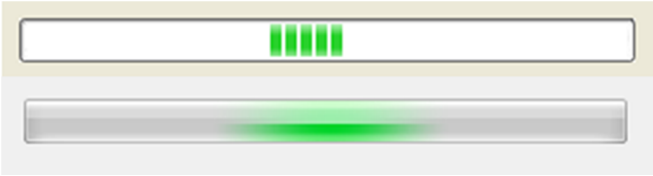


Figura 32 - Progressbar

As principais propriedades do controle são:

- MarqueeAnimationSpeed: velocidade da animação em milisegundos
- Maximun: maior tamanho da progressbar
- Minimun: menor tamanho da progressbar
- Style: estilo da progressbar

O principal evento do progressbar é o click que é acionado quando o usuário clicar com o mouse sobre o controle.

O controle Tooltip é associado a um controle e tem a função de mostrar alguma informação quando o cursor do mouse estiver sobre o controle.

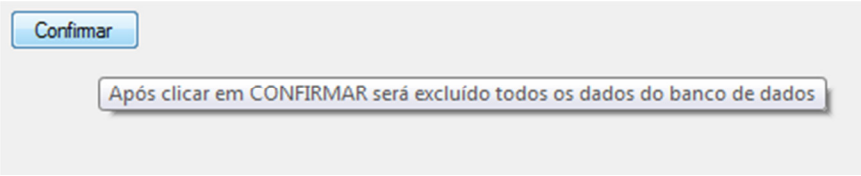


Figura 33 - Tooltip

A principal propriedade é o IsBalloon que indica se o texto estará no formato de balão.

Além disso, uma outra propriedade interessante são as propriedades de delay (initialdelay,automaticdelay,etc) que configura o delay(tempo) que o texto será exibido.

Depois de adicionado o controle tooltip no projeto, é só acessar um controle especifico e configurar a mensagem a ser exibida na propriedade Tooltip on Tooltip.

OBS 1:Essa configuração é feita na propriedade do controle que terá uma mensagem de informação.

OBS 2:Não é necessário adicionar novos controles tooltips para cada mensagem de informação que será exibida.

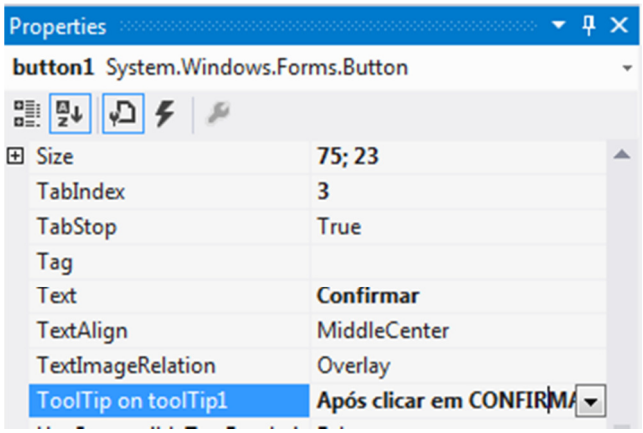


Figura 34 - Configurando a propriedade Tooltip em um controle

7.4. Menustrip, toolstrip e statusbartrip

Os menus são usados para fornecer grupos de comandos relacionados para aplicativos Windows.

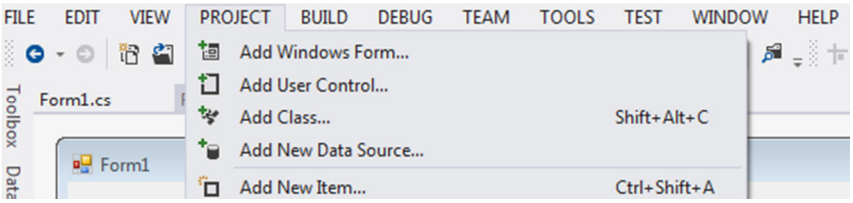


Figura 35 - Exemplo de menus

Um menu é formado por submenus(menus dentro de menus) e itens de menu. Os menus de nível superior aparecem na parte esquerda da figura, enquanto todos os submenus ou itens de menus são mostrados a direita.

Todos os itens de menu podem ter atalhos de tecla "Alt" e da letra sublinhada. Os menus que não são de nível superior também podem ter teclas de atalho(combinações de Ctrl, Shift,Alt,F1,F2,teclas de letra etc.)

Para criar um menu arraste um controle MenuStrip ao formulário.Para adicionar entradas no menu, clique na caixa de texto **TypeHere** e digite o texto que deve aparecer no menu. Cada entrada no menu é do tipo **MenuItem**. Após o programador pressionar a tecla **Enter**, o item de menu é adicionado. Então, mais caixas de texto **TypeHere** aparecem permitindo adicionar itens debaixo ou lado do item de menu original.

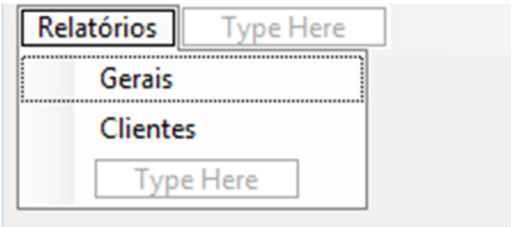


Figura 36 – Criando itens do menu

Para criar um atalho de acesso, digite um E comercial (&) na frente do caractere a ser sublinhado. Por exemplo, para criar o item de menu Arquivo, digite &Arquivo. Os programadores podem remover um item de menu selecionando-o com o mouse e pressionando a tecla DELETE.

Para inserir uma barra separadora adicione um item de menu e no texto do menu(**TYPE HERE**) digite "-" como texto do menu.

Podemos ainda colocar imagens em menus através da opção **SET IMAGE**

Clicando em **CONVERT TO** podemos converter um menu item para outro tipo: separator, menuitem, textbox..

Para inserir outros itens no menu, selecione a opção **INSERT**.

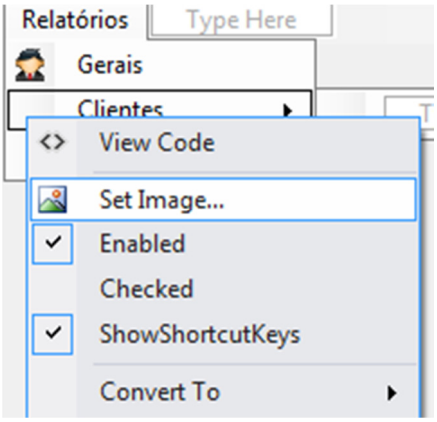


Figura 37 - Configurando os itens do menu

As principais propriedades do menu são:

- Checked: indica se um item de menu está marcado. O padrão é FALSE, que indica que o item de menu não está marcado
- Index: especifica a posição de um item em seu menu progenitor
- MenuItens: relaciona os itens de submenu de um item de menu específico
- RadioCheck: indica se um item de menu selecionado aparece como um botão de rádio ou exibe uma marca de visto.
- Shortcut: especifica a tecla de atalho para o item de menu
- Text: especifica o texto que aparece no item de menu

O principal evento do MenuStrip é o click que é gerado quando o item é clicado ou a tecla de atalho é usada. É o padrão quando se dá um clique duplo no projetista

O controle **ToolStrip** permite incluir uma barra de ferramentas no formulário. Para adicionar uma toolbar clique na guia Menus & Toolbars arraste e solte o controle **ToolStrip** no formulário.

Para adicionar itens na toolbar, clique no botão **Add ToolStripButton**. Clicando na opção acima podemos também escolher o tipo de item a ser adicionado na barra de ferramentas.

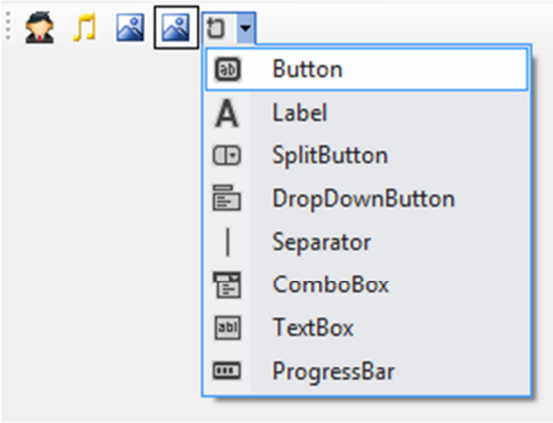


Figura 38 - Toolstrip

As principais propriedades da ToolStrip são:

- Items:Coleção de itens que compõem a caixa de ferramentas
- LayoutStyle: configura a orientação do layout da barra de ferramentas
- Podemos também alterar algumas configurações de um item da barra de ferramentas clicando com o botão direito do mouse sobre o item.
- A opção DisplayStyle altera o estilo de um item. Nessa opção podemos adicionar texto a um botão.
- A opção SetImage permite alterar uma imagem para o botão

O principal evento da ToolStrip é o evento click que é gerado quando o item é clicado

O controle **StatusStrip** é utilizado para apresentar informações ao usuário na parte inferior da janela de um programa.

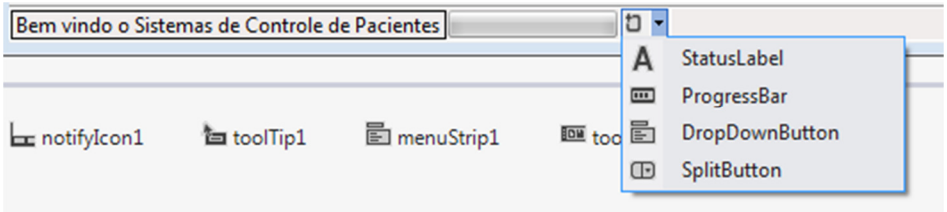


Figura 39 - Statustrip

Para adicionar um controle **StatusStrip** clique na guia **Menus & Toolbars** arraste e solte o controle **StatusStrip** no formulário.

Para adicionar itens na barra de status do formulário, clique no botão **Add ToolStripStatusLabel.**

As principais propriedades da StatusStrip são:

- Items:Coleção de itens que compõem a caixa de ferramentas
- LayoutStyle: configura a orientação do layout da barra de ferramentas

O principal evento da ToolStrip é o evento click que é gerado quando um item da ToolStrip é clicado.

Assim como os demais controles de menus, podemos alterar algumas configurações de um item da barra de ferramentas clicando com o botão direito do mouse sobre o item.

Um menu de contexto é geralmente um menu suspenso que aparece quando você clica com o botão direito do mouse sobre um controle , uma janela , etc..

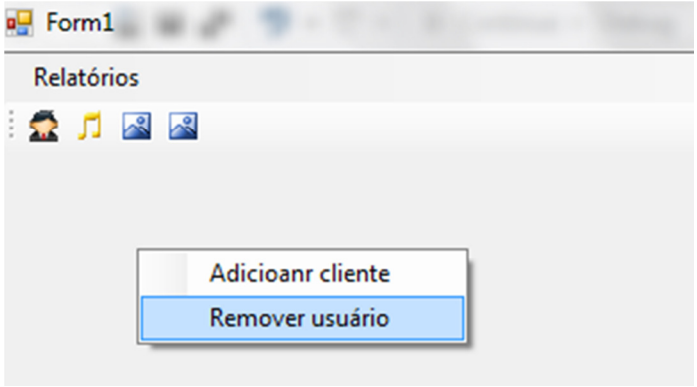


Figura 40 - Contextmenu

Para adicionar um menu de contexto insira o controle - ContextMenuStrip que está na guia Menus & Toolbars. Em seguida digitaremos diretamente no componente incluído do formulário as opções do seu menu de contexto.

Depois de configurar o menu de contexto podemos associar o componente **ContextMenu** a um controle definindo a propriedade **ContextMenu** do controle

ao nome do **ContextMenu**. Na imagem abaixo é configurado a propriedade ContextMenu de um formulário.

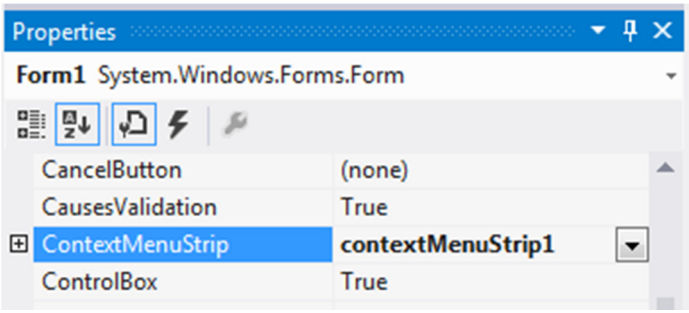


Figura 41 - Configurando o contextmenu de um formulário

As propriedades e eventos do menu de contexto são semelhantes ao do MenuStrip

Para abrir um formulário a partir de um item de Menu, devemos programar o evento Click do menu para carregar um formulário especifico. Veja como ficaria o código

```
    private void adicioanarClienteToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Form2 formCliente = new Form2();
            formCliente.Show();
        }
```

No código acima foi criado um objeto formCliente do tipo Form2. Em seguida através do método Show o formulário será carregado na tela.

8. Acesso a dados

Para iniciar o acesso a dados com o Visual C# o primeiro passo é definir qual o SGBD que será utilizado e também a biblioteca de dados, que dependerá do SGBD definido.

Exemplos:

- Access: OleDb
- SQL Server: SqlClient
- My SQL: MySQLClient

No caso do MySQL, Postgres e outros SGBD que não são nativos da plataforma .Net, será necessário fazer o download do conector ao SGBD. Após a instalação do conector específico, o acesso a dados é realizado da mesma forma que será mostrado nesta apostila.

Para apresentar o acesso a dados, iremos realizar um cadastro de contato que será composto dos seguintes dados:

- Codigo (Chave Primária)
- Nome
- E-mail
- Celular

Após a criação do banco de dados e da tabela contato no SQL Server, será necessário criar um formulário para cada campo da tabela Contato. Como o campo código é uma chave primária e será gerado automaticamente, não é necessário criar esse campo no formulário.

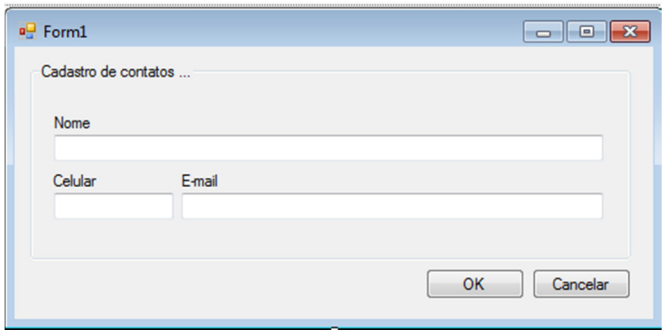


Figura 42 - Cadastro de contatos

Neste exemplo apresentado na apostila, será mostrado o acesso de dados de forma manual, ou seja, escrevendo linha por linha de código. O Visual C# pode facilitar este acesso, utilizando controles disponíveis na toolbox -> data, mas para melhor compreensão dos mecanismos de acesso a dados, utilizaremos apenas linha de comando para apresentação. Na primeira linha de código da aplicação devemos importar a biblioteca a ser utilizada para acesso a dados, neste caso o SQL Client.

```
using System.Data.SqlClient;
```

Em seguida devemos incluir os seguintes objetos de acesso a dados:

- O primeira que será uma objeto do tipo **Connection**, que armazenará as informações referente a string de conexão
- O segundo será do tipo **Command** que será responsável pela execução do comando SQL.

OBS: Estes objetos deverão ser criados no início da classe do formulário, para que todos os eventos criados possam utilizar estes objetos, que nestes casos são GLOBAIS para o formulário ao qual eles foram criados.

```
SqlConnection conexao = new SqlConnection("string de conexão do banco de dados");

SqlCommand comando = new SqlCommand();
```

A string de conexão do banco de dados, mostrado no objeto SQLConnection (ou connection string) é um código utilizado pelo sistema para se conectar ao banco de dados. Cada banco de dados tem a sua própria string. No SQL Server, a string de conexão tem o seguinte formato:

```
Data Source=Endereço do Servidor SQL Server;Initial Catalog=nome do banco de dados;User Id=nome do usuário do banco de dados;Password=senha do usuário;
```

Em seguida vamos programar o evento Click() do botão OK que irá inserir os valores informados nas caixas de texto do formulário nos respectivos campos da tabela Contato.

O código de acesso a dados ficará dentro de uma estrutura de tratamento de exceção chamada: **try...catch...finally**

Dentro da estrutura **try** ficará o código que insere o registro no banco de dados.

Dentro da estrutura **catch** ficará o código caso algum erro aconteça nos comandos presente na estrutura **try**

Por último ainda existe **finally** que independente de erro ou não será sempre executado.

Primeiro vamos atribuir ao atributo comando o atributo conexão. Dessa forma o comando SQL presente no atributo COMANDO, será executado no banco de dados informado no atributo CONEXAO.

```
        comando.Connection = conexao;
```

Depois devemos atribuir qual o comando SQL a ser executado. Veja no exemplo abaixo onde é representada a estrutura para o comando INSERT.

```
    comando.CommandText = "INSERT INTO Contato(Nome,Celular,Email)";
    comando.CommandText += "values ('" + txtNome.Text + "', '" + txtCelular.Text +
"',";
    comando.CommandText += "'" + txtEmail.Text + "')";
```

No código mostrado acima, observe o seguinte:

```
        INSERT INTO Contato(Nome,Celular,Email)
```

O trecho representa a primeira parte do comando SQL para inserção, onde está sendo inserido os campos Nome,Celular,Email na tabela Contato

```
        values ('" + txtNome.Text + "', '" + txtCelular.Text + "',
'" + txtEmail.Text + "')
```

O trecho acima representa os valores que serão inseridos, no qual estão separados pela sintaxe '" + campo do formulário + "'

O próximo passo é composto de 3 operações

- Abrir o banco de dados
- Executar o comando
- Fechar o banco de dados

Abaixo veja o código para realizar essas operações:

```
conexao.Open();
comando.ExecuteNonQuery();
conexao.Close();
```

O código representado acima é válido para os comandos de:

- Inserir: INSERT
- Atualizar: UPDATE
- Excluir: DELETE

Veja abaixo um exemplo para atualizar um registro no banco de dados a partir do Visual Basic:

```
     comando.CommandText = "UPDATE Contato set Nome  = '" +
txtNome.Text + "' where Codigo = '" + Convert.ToInt32(codigo) +
"'
```

Veja abaixo o comando para excluir um registro a partir do Visual Basic

```
     comando.CommandText = "Delete From Contato where Codigo =
'" + Convert.ToInt32(codigo) + "'
```

Para fazer as operações de CONSULTA ou SELECT, será necessário além um objeto Conexao, um objeto do tipo Adapter e um outro objeto do tipo dataset

O objeto Adapter é o responsável por executar um comando de consulta, e o objeto DataSet é o responsável por guardar os valores retornados

Além dos objetos de conexão será necessário um componente visual chamado **DataGridView** que é responsável por apresentar os dados que foram retornados

pela Consulta ao Banco de Dados. O componente DataGridView está localizado na toolbar -> data. Veja abaixo como ficará o formulário.

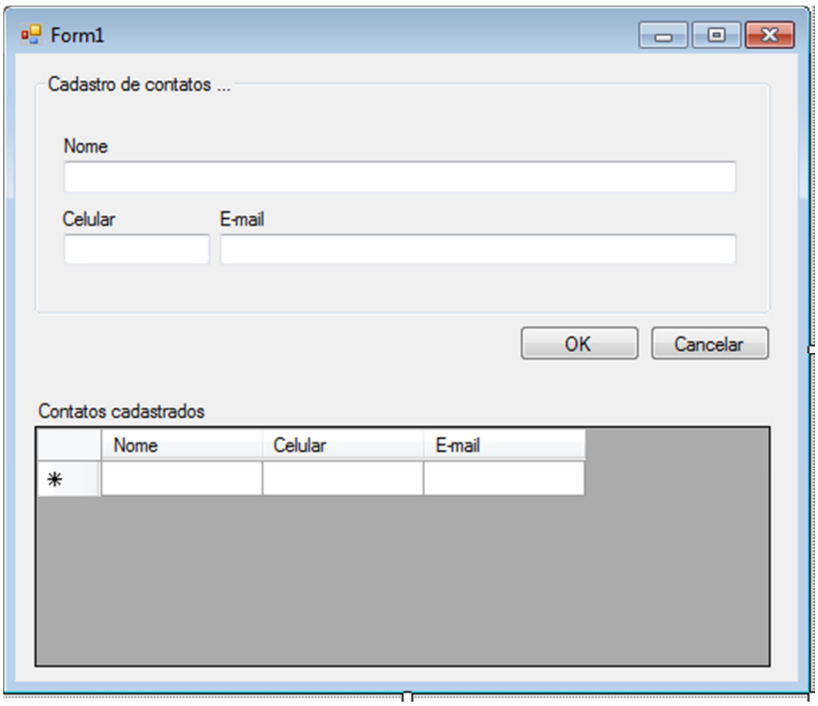


Figura 43 - Formulário com Datagridview

Para preencher o datagridview criaremos uma função que preenche o datagridview com todos os registros da tabela contato. Antes de codificar a função, devemos declarar mais dois objetos: um **dataset** e um **adapter**. O **dataset** e o **adapter** deverão está declarados no mesmo local dos objetos connection e command, para que também sejam GLOBAIS ao formulário.

A declaração do objeto adapter e do objeto dataset deve ser como apresentado abaixo:

```
SqlDataAdapter adapter = new SqlDataAdapter();

DataSet ds = new DataSet();
```

A função **preencheGrid** deverá ser criada conforme o código a seguir. A decisão de criar a função **preencheGrid** é pela razão de em qualquer necessidade de preencher o grid, é só realizar a chamada desta função, não sendo necessário criar novo código para cada nova visualização da Grid.

```
     private void preencheGrid()
        {
            ds = new DataSet();
            adapter = new SqlDataAdapter("SELECT Nome,Celular,Email from
Contato", conexao);
            adapter.Fill(ds, "Contato");
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Contato";
        }
```

A função criada tem a responsabilidade de preencher o **dataset** com a consulta SQL presente no adapter. Em seguida é atualizada as propriedades **DataSource** e **DataMember** do dataset

- DataSource: fonte de dados(dataset)
- DataMember: tabela do banco de dados(contato)

Depois de criada a função é chamada no evento Load do formulário e no evento click do botão OK, para que a cada novo contato cadastrado, o grid seja atualizado.

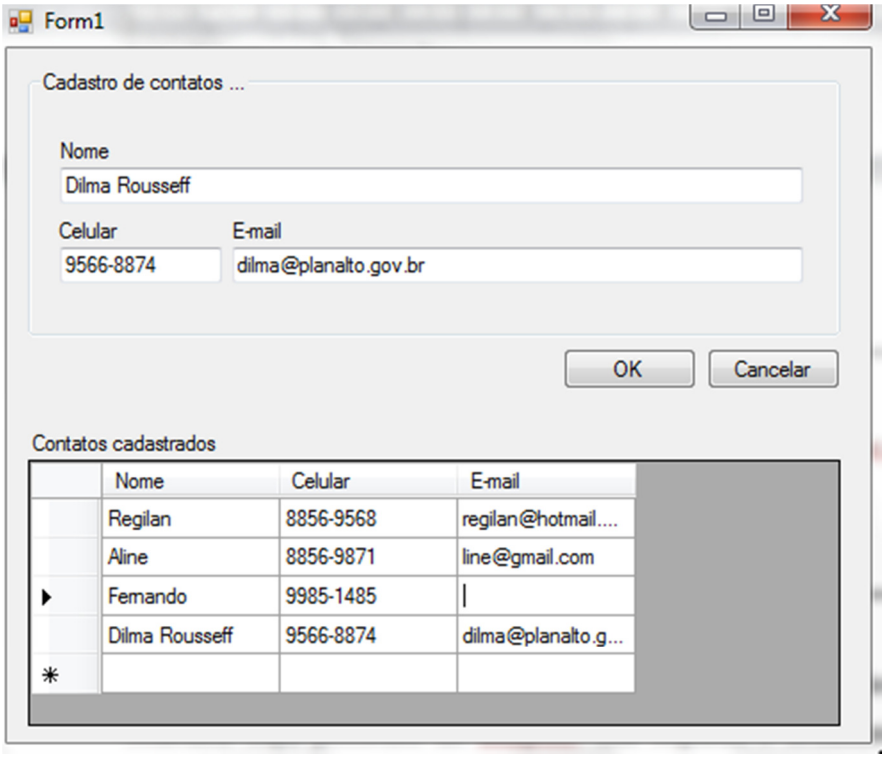


Figura 44 - Formulário de cadastro de contato

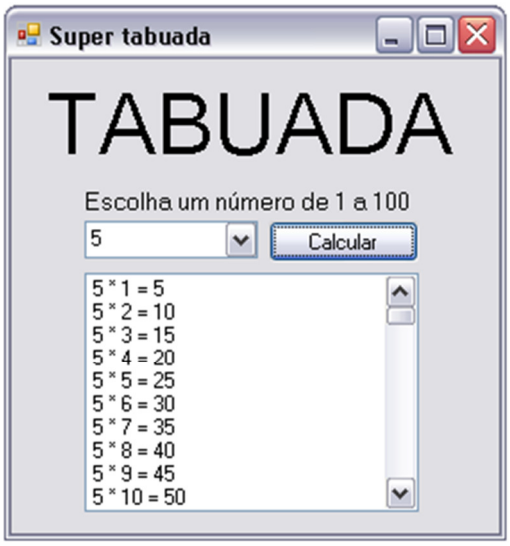
O código completo da aplicação é mostrado a seguir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Data.SqlClient;


namespace WindowsFormsApplication4
{
    public partial class Form1 : Form
    {

        SqlConnection conexao = new SqlConnection("string de conexão do banco de
dados");

        SqlCommand comando = new SqlCommand();

        SqlDataAdapter adapter = new SqlDataAdapter();

        DataSet ds = new DataSet();


        private void preencheGrid()
        {
            ds = new DataSet();
            adapter = new SqlDataAdapter("SELECT Nome,Celular,Email from
Contato", conexao);
            adapter.Fill(ds, "Contato");
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Contato";
        }


        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            preencheGrid();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            comando.Connection = conexao;
            comando.CommandText = "INSERT INTO Contato(Nome,Celular,Email)";
            comando.CommandText += "values ('" + txtNome.Text + "', '" +
txtCelular.Text + "',";
            comando.CommandText += "'" + txtEmail.Text + "')";

            conexao.Open();
            comando.ExecuteNonQuery();
            conexao.Close();

            preencheGrid();
```

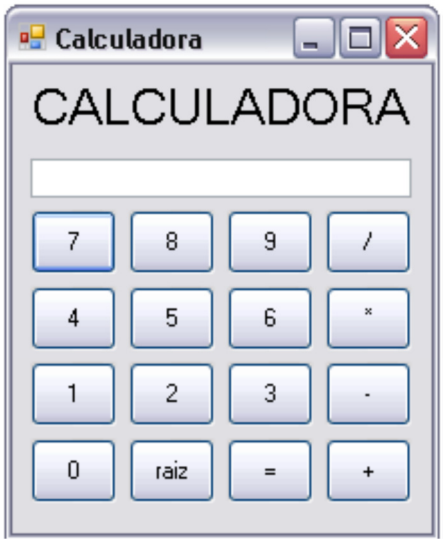
```
        }
    }
}
```

9. Extras: Exercícios propostos para construção de interfaces gráficas

Questão 01. Desenvolver um aplicativo em C# com a finalidade de calcular a tabuada dos números de 1 a 100. Utilize como exemplo para construção da interface gráfica o modelo abaixo. Nesse exemplo, o usuário escolheu o número 5 e após clicar no botão Calcular, o programa gerou a tabuada de 1 a 100 do número 5.



Questão 02. Com base no modelo de calculadora abaixo, você deverá programar as 4 operações básicas (adição, subtração, multiplicação e divisão), raiz quadrada e adicionar as seguintes funções:



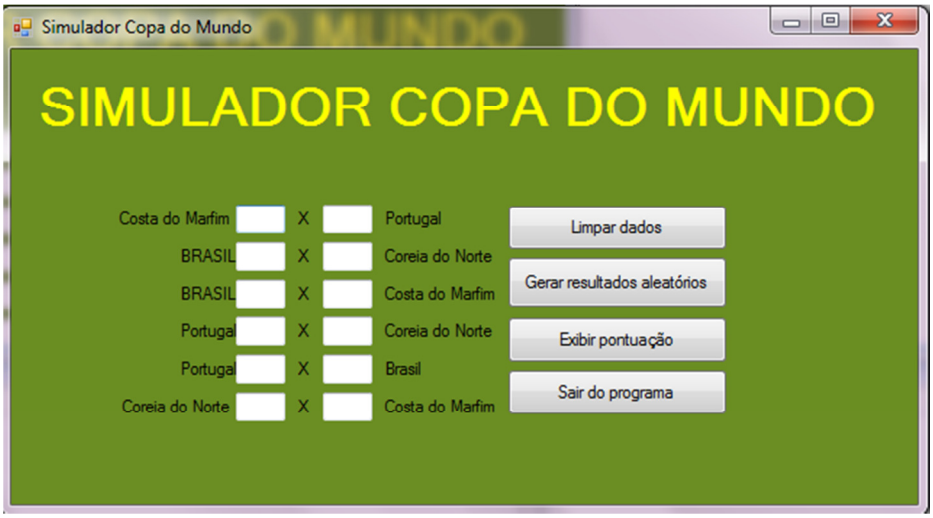
a) Seno
b) Cosseno
c) Tangente
d) Potência (Elevar um número pelo outro)
e) Fatorial
f) Converter de graus para radianos

Questão 03. Desenvolva o formulário abaixo e em seguida escreva o código para gerar as funcionalidades indicadas.

- Botão Adicionar Categoria: adiciona uma categoria no cardápio (Exemplo: Massas, Filet,Drinks,Peixes)
- Botão Adicionar Refeição: adiciona uma refeição a uma categoria(Exemplo: Pizzas, Spaghetti na categoria Massas)
- Botão Excluir: Exclui um registro da treeview
- Botão OK: Exibe uma mensagem com o texto: Restaurante Toscana – A Melhor Opção A La Carte da Cidade
- Botão Cancelar: Fecha o formulário

Questão 04. Desenvolver um aplicativo em C#, com a finalidade de simular e exibir a pontuação das seleções de um determinado grupo da Copa do Mundo 2014 no Brasil.

O aplicativo deverá ter funcionamento semelhante à interface abaixo:



- Ao clicar no botão Limpar Dados, os registros deverão ser apagados de forma que o usuário possa gerar novas simulações.
- Ao clicar no botão Gerar resultados aleatórios, seu programa deverá utilizar a classe Random para gerar os placares dos jogos de forma aleatória. OBS: Considere como valor mínimo 0 e valor máximo 8)
- Ao clicar no botão Exibir Pontuação, o programa deverá exibir uma Caixa de Mensagem com a pontuação das seleções.
- Ao clicar no botão Sair, o programa deverá ser fechado.

Questão 05. Um analista de sistema projetou um programa que será instalado em uma Transportadora. Após o estudo sobre o projeto, o analista constatou que o sistema possui as seguintes funções.

Cadastros Gerais	Manutenção e Serviços	Ocorrências	Registros de Viagens	Funcionários
Cliente	Peças	Acidentes	Chegada	Férias
Funcionário	Serviços Gerais	Roubo de Mercadorias	Saída	Reajuste Salarial
Motoristas	Troca de Pneus	Outras Ocorrências	Ocorrências	Acompanhamento Médico
Fornecedores	Troca de Óleo		Entrega de Encomendas	Currículo
Oficinas	Abastecimentos		Diárias de Motoristas	Banco de Talentos
Veículos				
Postos				
Rotas				

Diante da tabela acima, crie um novo projeto no Visual C# com um MENUSTRIP, de forma que o usuário do sistema possa acessar da melhor forma possível as funções disponibilizadas pelo software.

Adicione também uma barra de ferramentas com algumas opções do menu e uma barra de status com uma mensagem de boas vindas ao usuário conforme abaixo. (As mensagens deverão ser customizadas de acordo com o horário de acesso ao sistema)

- Bom Dia caro usuário, Bem vindo ao Sistema da Transportadora Chega Logo!
- Boa Tarde caro usuário, Bem vindo ao Sistema da Transportadora Chega Logo!
- Boa noite caro usuário, Bem Vindo ao Sistema da Transportadora Chega Logo!

Por último configure o menu principal para ser acionado quando o usuário usar a tecla de atalho ALT. Adicione também ShortcutKey para alguns itens dos menu.

Questão 06. Criptografia é o estudo dos princípios e técnicas pelas quais a informação pode ser transformada da sua forma original para outra ilegível, de forma que possa ser conhecida apenas por seu destinatário (detentor da "chave secreta"), o que a torna difícil de ser lida por alguém não autorizado. Assim sendo, só o receptor da mensagem pode ler a informação com facilidade.

Sendo assim você deverá desenvolver um programa com interface gráfica para criptografar uma frase de no máximo 100 caracteres e em seguida mostrar essa frase criptografada ao usuário. Faça também uma função para descriptografar uma frase com base no algoritmo de criptografia desenvolvido. Veja o exemplo abaixo como poderia ser uma criptografia.

EXEMPLO:

A Vida é Bela = J 8kop m Om7p

J 8kop m Om7p = A Vida é Bela