

Colégio Técnico Industrial “Prof. Isaac Portal Roldán”

Técnico em Informática – 2º ano



Microsoft Visual Studio .Net - C#

Prof. André Luiz Ribeiro Bicudo





Unidade I

Carga Horária Prevista

- Três aulas - 1ª Semana

Objetivos

- Conhecer os tipos de ferramentas utilizadas para o desenvolvimento de aplicações.
- Compreender a estrutura da ferramenta Microsoft Visual Studio .NET C# 2019.
- Instalar a ferramenta.
- Desenvolver um primeiro projeto exemplo.

Conteúdos

- IDE (Integrated Development Environment).
- RAD (Rapid Application Development).
- Microsoft Visual Studio .NET C# 2019.
- .NET Framework.
- Instalando Microsoft Visual Studio .NET C# 2019.

1. INTRODUÇÃO

Na primeira unidade desta disciplina, você terá a oportunidade de conhecer os conceitos referentes ao IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado) e ao RAD (Rapid Application Development ou Desenvolvimento Rápido de Aplicação).

Além disso, será apresentado um breve comentário sobre o pacote de ferramentas da Microsoft e o Visual Studio .NET C# 2019 Versão Community. Para finalizar o estudo desta unidade, será apresentada a ferramenta Microsoft Visual Studio .NET C# 2019 - Community, considerada o foco de nossa disciplina.

2. INTRODUÇÃO AO MICROSOFT .NET

O principal objetivo do Microsoft .Net é simplificar o desenvolvimento de aplicações para Windows Desktop, Web, aplicações customizadas e serviços. Isto inclui aplicações Mobile, aplicações hospedadas na nuvem e acesso a aplicações com servidores de Banco de Dados, dentre muitas outras possibilidades.

O Microsoft .Net Framework é a infraestrutura do Microsoft .Net, ele é um modelo de programação orientado a objetos consistente e de fácil entendimento que proporciona um ambiente para desenvolvimento de aplicações e serviços independente de plataforma ou dispositivos.

Implementa infraestrutura para integrar, executar, operar e gerenciar todas as soluções baseadas em servidores, como: Windows Server, Microsoft Share Point, SQL Server, servidores WEB rodando IIS (Internet Information Services), dentre outros serviços oferecidos pela Microsoft.

3. MICROSOFT VISUAL STUDIO

O Microsoft Visual Studio constitui o core do desenvolvimento do Microsoft .Net. É um ambiente de desenvolvimento completo no qual você pode fazer o design, desenvolvimento, debug e distribuição de suas aplicações e serviços. Permite criar aplicativos Web ASP.NET, XML Web services, aplicativos Windows Desktop e Console, aplicativos móveis, desenvolvimento de games, dentre outros.

Dentre muitas linguagens de programação suportadas pelo .Net, temos: Visual Basic, Visual C#, Visual C++, Visual J#, Visual Web Developer, dentre outras.

Essas ferramentas utilizam o mesmo ambiente de desenvolvimento facilitando a criação de aplicações, além de aproveitar as funcionalidades do .NET Framework, que fornece acesso a tecnologias que simplificam o desenvolvimento de aplicações Web ASP e XML Web Services.

A distribuição Express do Visual Studio é gratuita, inclusive para a utilização comercial. Portanto, não será necessário o pagamento de licença para o uso de qualquer ferramenta desse pacote na distribuição Express.

4. INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

Integrated Development Environment (IDE) ou Ambiente de Desenvolvimento Integrado é um programa de computador que reúne componentes que apoiam o desenvolvimento de aplicações com o objetivo de agilizar este processo. Estes componentes são:

- **editor:** edita o código-fonte do programa com a(s) linguagem(ns) de programação suportada(s) pelo IDE;
- **compilador/montador:** compila o código-fonte do programa, transformando-o em linguagem de máquina e, em seguida, agrupa estes programas compilados em um arquivo executável.
- **debugador:** auxilia no processo de encontrar e corrigir erros (bugs) no código-fonte do programa.

5. RAPID APPLICATION DEVELOPMENT (RAD)

A evolução dos Sistemas Operacionais com Interface Gráfica fez com que as tarefas relacionadas ao desenvolvimento de aplicações sofressem adaptações.

Embora os desenvolvedores de sistemas já usassem IDEs, eles sentiam necessidade de utilizar ferramentas que permitissem agilizar ainda mais o desenvolvimento, que possuísssem funcionalidades para suprir a demanda de criação de interfaces, modelagem, testes e manutenção das aplicações. A partir deste contexto, começaram a surgir as primeiras IDEs que uniram as técnicas de RAD (Rapid Application Development ou Desenvolvimento Rápido de Aplicação) com o objetivo de associar, de maneira simples e rápida, um elemento de interface e o código da aplicação.

Entretanto, com a evolução dessas IDEs e das técnicas de RAD, mais funcionalidades foram inseridas. Atualmente, estas ferramentas são compostas, além dos componentes básicos das IDEs, por:

- **Design de interface:** criação de interfaces (telas) da aplicação com cliques do mouse, por meio da inserção de componentes pré-definidos.

(1) Templates : são modelos de programação que implementam funcionalidades que são manipuladas a partir da parametrização de rotinas e funções.

- **Modelagem:** criação do modelo de classes, objetos, interfaces, associações e interações dos artefatos envolvidos no software com o objetivo de solucionar as necessidades da aplicação.
- **Geração de código:** a geração de código também é encontrada em IDEs, contudo com um escopo mais direcionado a **templates**¹ de código comumente utilizados solucionar problemas rotineiros. Todavia, em conjunto com ferramenta de modelagem, a geração pode gerar todo ou praticamente todo o código-fonte do programa baseado no modelo proposto, tornando mais rápido o processo de desenvolvimento e distribuição do software.
- **Distribuição:** auxilia no processo de criação do instalador do software ou outra forma de distribuição deste, seja através de discos ou via internet.
- **Testes automatizados:** realiza testes no software de forma automatizada, com base em programas de testes previamente especificados; gera um relatório e, assim, auxilia na análise do impacto das alterações no código-fonte.
- **Refatoração:** consiste na melhoria constante do código-fonte da aplicação, como por exemplo, a construção de código mais otimizado.

Diagrama de um projeto em C#

O diagrama a seguir ilustra os relacionamentos entre o tempo de compilação e o tempo de execução dos arquivos de código fonte C#, as bibliotecas de classes base, módulos (assemblies), e o CLR.

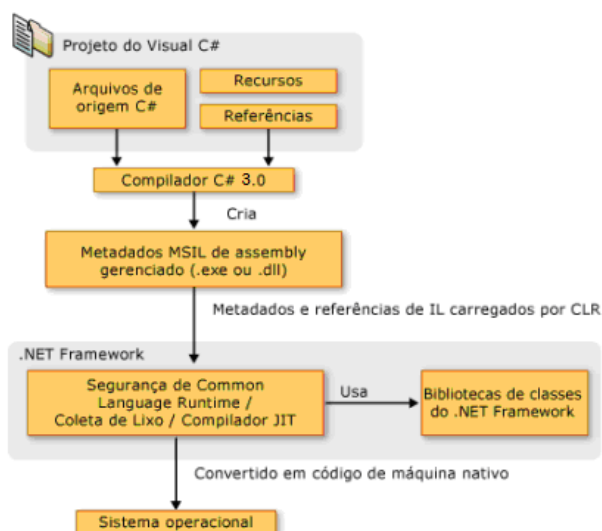


Figura 1: Diagrama de um projeto em C#

AMBIENTE DE DESENVOLVIMENTO INTEGRADO MICROSOFT VISUAL STUDIO

O Microsoft Visual C# 2019 é um ambiente de desenvolvimento integrado (IDE), que também se enquadra como uma ferramenta RAD (Rapid Application Development). Esta ferramenta foi desenvolvida pela Microsoft com o intuito de proporcionar facilidade aos desenvolvedores de sistemas que desejam criar aplicações, oferecendo, desta forma, uma vasta quantidade de recursos para este desenvolvimento.

O C# é uma linguagem de programação orientada a objetos e faz parte da plataforma .Net. Essa linguagem tem a sintaxe e estruturas básicas de programação com base na linguagem C++ e assemelha-se ao Java.

Instalando Microsoft Visual Studio 2019 Community

Para instalar o Microsoft Visual C# 2019 Community, além de poder obter mais informações como cursos e materiais, acesse o link:

<https://www.visualstudio.com/pt-br/vs/community/>

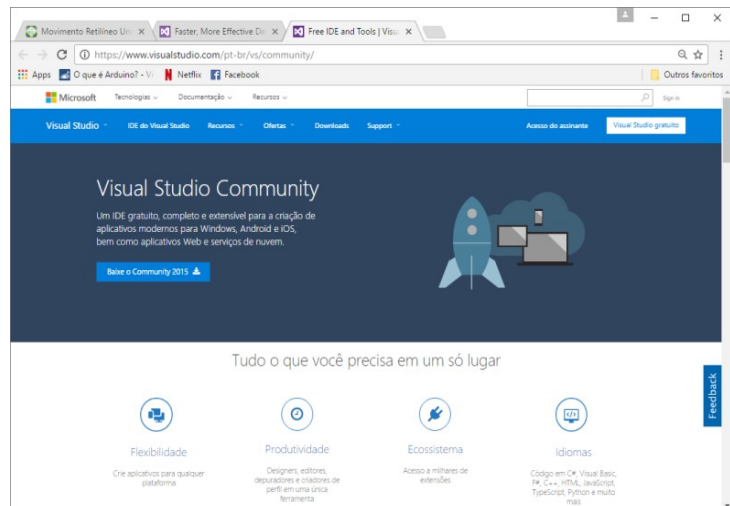


Figura 2: Site para download do Visual Studio

Iniciando um novo projeto

Através do menu File, opção New Project – Project será apresentada a tela ao lado com todas as opções de linguagens, ferramentas e tipos de templates de projetos.

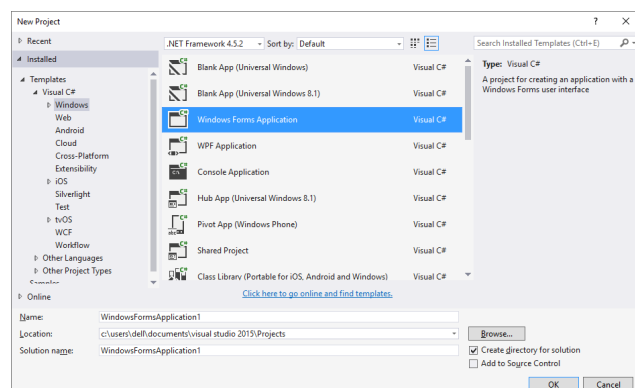


Figura 3: Escolha do tipo de projeto

6. E-REFERÊNCIA

Estrutura do .Net Framework: Disponível em:

[https://msdn.microsoft.com/pt-br/library/zw4w595w\(v=vs.110\).aspx](https://msdn.microsoft.com/pt-br/library/zw4w595w(v=vs.110).aspx). Acesso em: 3 fevereiro 2018.

Página de download Visual Studio Community: Disponível em:

<https://www.visualstudio.com/pt-br/vs/community/>. Acesso em: 3 fevereiro 2020.

MSDN Brasil. Disponível em: <https://msdn.microsoft.com/pt-br/default.aspx>. Acesso em: 3 fevereiro 2018.

Site oficial da Microsoft sobre o Visual Studio. Disponível em: <https://www.visualstudio.com/>. Acesso em: 3 fevereiro 2020.

Site oficial Microsoft sobre .Net. Disponível em <https://www.microsoft.com/net/learn>. Acesso em: 3 de fevereiro de 2020.



Unidade II

Carga Horária Prevista

- Seis aulas - 2ª e 3ª Semana

Objetivos

- Aprender a utilizar a ferramenta de desenvolvimento Microsoft Visual C# 2019, seu ambiente de trabalho, seus recursos e suas funcionalidades.
- Conceituar objetos, propriedades, eventos e métodos.
- Desenvolver uma aplicação baseada na ferramenta Microsoft Visual C# 2019.

Conteúdos

- Ambiente de Trabalho do Microsoft Visual C# 2019.
- Objetos.
- Formulários.
- Controles.
- Propriedades.
- Métodos.
- Eventos.
- Criando uma aplicação no Microsoft Visual C# 2019.

7. INTRODUÇÃO

Na unidade anterior, você conheceu os conceitos referentes ao IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado) e ao RAD (Rapid Application Development ou Desenvolvimento Rápido de Aplicação), além de aprender os procedimentos para instalar o Microsoft Studio 2019..

Nesta unidade será apresentado para você o Ambiente de Desenvolvimento Integrado Microsoft Visual Studio 2019, por meio do qual será possível manipular projetos e formulários, inserir controles e codificar as ações que serão criadas em sua aplicação.

8. AMBIENTE DE TRABALHO DO MICROSOFT VISUAL STUDIO

O Ambiente de Desenvolvimento Integrado Microsoft Visual Studio possui um conjunto de ferramentas integradas para construir aplicações para Windows com rapidez e eficiência. Originalmente ele está disposto de acordo com a Figura 3.

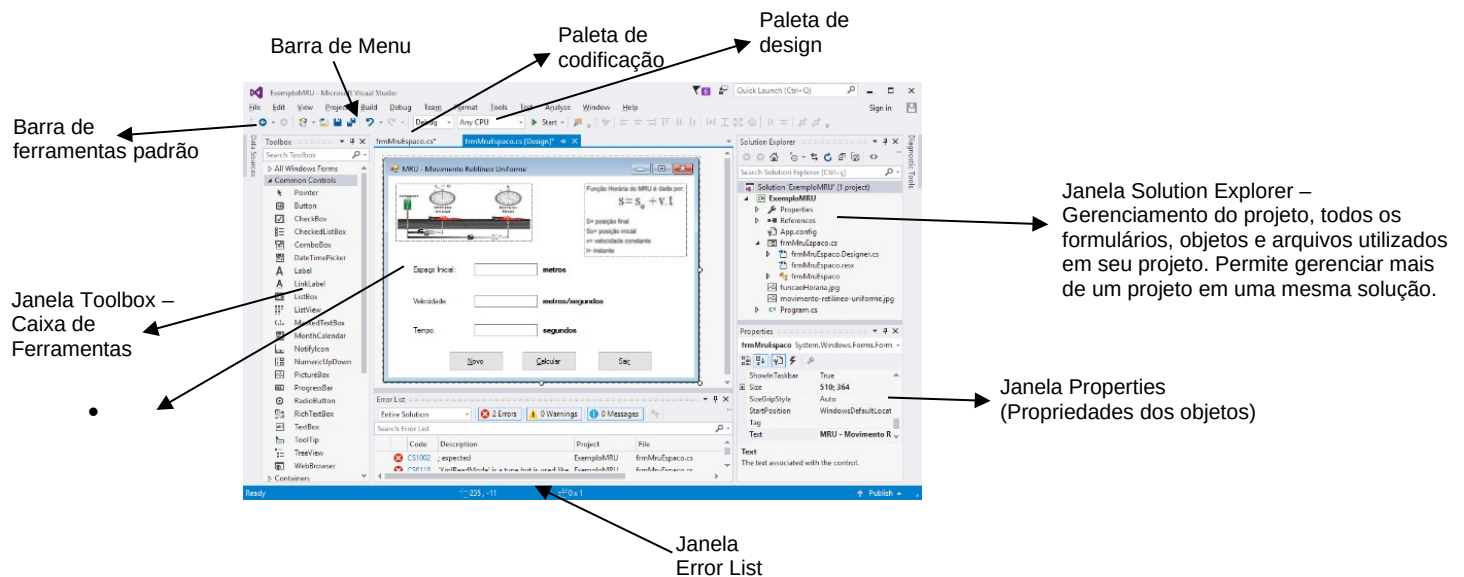


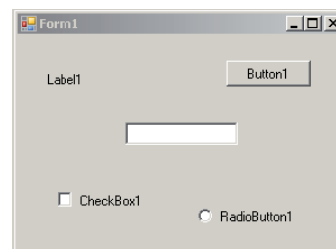
Figura 4: Interface de trabalho do Microsoft Visual C# 2019.

- Barra de menus: concentra todas as funcionalidades que o ambiente dispõe para o desenvolvimento da aplicação.
- Barra de ferramentas padrão: agrupa algumas funcionalidades disponíveis na Barra de menus, facilitando sua utilização. Exemplo: novo projeto, novo item, salvar tudo, executar etc.
- Janela Toolbox: contém os elementos da interface de usuário (controles) que podem ser inseridos no formulário.
- Janela Properties: concentra as características do formulário e dos elementos de interface de usuário (controles) inseridos no formulário.
- Janela Solution Explorer: apresenta a estrutura do projeto organizada por tipos (projeto, formulários, módulos etc.).
- Janela Error List: apresenta a relação de erros ocorridos na aplicação.
- Paleta de Design: área criada para a inserção dos elementos de interface de usuário (controles) no formulário criado anteriormente.
- Paleta de codificação: área reservada para a inserção da codificação do formulário e dos controles existentes nele.

Objetos

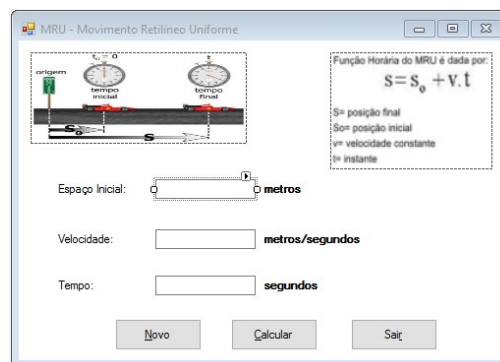
Os objetos são representações que compõem o projeto da aplicação que é desenvolvida no Ambiente de Desenvolvimento Integrado Microsoft Visual C#, como é o caso, por exemplo, dos formulários e controles. Qualquer controle, formulário, relatório, etc., se enquadram como objetos.

Todo objeto tem propriedades, métodos e eventos agregados, por meio dos quais é possível definir suas características e utilizar suas funcionalidades e ações.

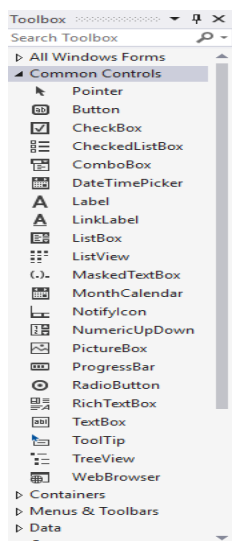


Formulários

Basicamente, quase todas as janelas apresentadas em uma aplicação são formulários. Eles servem de base para a inserção de controles e associam a programação dos eventos e lógica necessária para o seu funcionamento.



Controles (Objetos - Componentes)



Os controles são elementos pertencentes à interface do usuário que, depois de acrescentados ao formulário, se tornam objetos que poderão ser programados para funcionalidades específicas. Quando a aplicação for executada, os controles ficarão visíveis ao usuário e funcionarão como objetos padronizados como os de qualquer aplicação no padrão Windows.

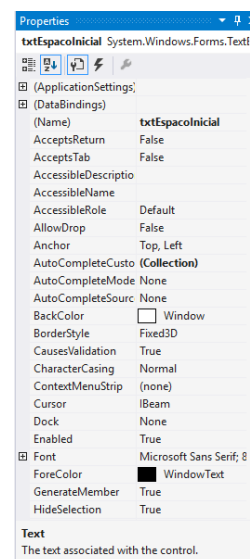
Os controles apresentados na ToolBox são controles nativos do Visual C#. Para adicionar novos controles, clique com o botão direito na ToolBox e escolha a opção Choose Items.

Propriedades

Propriedades são atributos que definem as características para cada objeto.

Como exemplo, pense em um objeto “motocicleta”. Uma propriedade para este objeto pode ser “cor vermelha”.

No Visual C#, as propriedades de um formulário, por exemplo, são definidas a partir da janela Properties. A propriedade Text desse objeto define o título da janela formulário. No exemplo ao lado, são apresentadas as propriedades de uma Caixa de Texto (Text Box), a propriedade Name (txtEspacoInicial) é fundamental, pois através dela o objeto é referenciado na programação, para acessar suas propriedades, ativar métodos e programação de eventos associados a este objeto.



do

Métodos

Métodos são funcionalidades agregadas a objetos. Como exemplo, siga o raciocínio do item anterior (propriedades). Imagine um objeto “motocicleta”, que tem como propriedade a “cor vermelha” e, além disso, pode ter como método as funcionalidades “acelerar” e “frear”.

No Visual C#, por exemplo, podemos chamar o método Focus, fazendo com que o foco do Windows seja direcionado a este objeto.

```
btnCalcular.Focus();
```

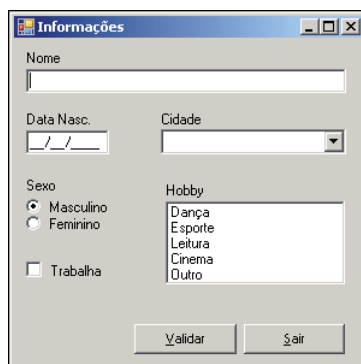
Eventos

Os eventos são ações atribuídas aos objetos. Pensando novamente no exemplo do objeto “motocicleta”, o qual tem atribuída a “cor vermelha” como propriedade, “acelerar” e “frear” como métodos. Como eventos podem ser citados “ao bater” e “ao ligar”.

No Visual C#, podem ser citados como exemplos de eventos aqueles relacionados ao formulário. Se for escolhido o evento Load, ele será executado assim que o formulário for carregado na memória em tempo de execução.

9. CRIANDO NOSSA PRIMEIRA APLICAÇÃO C#

Para esta primeira aplicação, vamos criar um formulário para a digitação de informações de uma pessoa, conforme layout apresentado abaixo:



Informações

Nome:

Data Nasc.: / / Cidade:

Sexo: ☒ Masculino ☐ Feminino ☐ Trabalha

Hobby:

Dança
Esporte
Leitura
Cinema
Outro

10. E-REFERÊNCIA

Site para desenvolvedores - Disponível em: www.linhadecodigo.com.br. Acesso em: 3 fevereiro 2018.



Unidade III

Carga Horária Prevista

- Três aulas

Objetivos

- Utilizar variáveis e definir os tipos de dados.
- Identificar as Estruturas de Controle de Programação.
- Utilizar os Métodos para conversão de tipos de dados.

Conteúdos

- Variáveis
- Tipos de Dados
- Estruturas de Controle
- Métodos para conversão de tipos de dados

11. INTRODUÇÃO

Na unidade anterior, desenvolvemos os primeiros projetos utilizando os formulários e objetos, bem como a programação de alguns eventos.

Nesta unidade veremos a correta utilização de variáveis, definir os tipos de dados, estruturas de controle e funções de conversão, que são necessários para o desenvolvimento da programação em C#.

12. VARIÁVEIS

As variáveis correspondem a posições (loais) na memória RAM do computador que receberão os dados a serem tratados em seu programa. Toda variável possui a identificação do tipo de dado que ela armazenará, um nome, um tamanho e um valor. No C#, todas as variáveis devem ser declaradas, podendo ser feito a declaração no início do programa ou no local onde você for utilizar a variável pela primeira vez. Quando você for declarar uma variável utilize a sintaxe:

```
tipo nomeVariável;      ou      tipo nomeVariável = valorInicial;
```

13. TIPOS DE DADOS

No C#, os tipos de dados se dividem em **value types** e **reference types**. No grupo dos value types temos char, int, float, enum, struct, etc., enquanto que no grupo definido como reference types temos classes, interface, delegate e array.

A seguir veremos exemplos dos principais tipos de dados existentes em C#.

Char

Representa um único caractere.

```
char sexo = 'F';  
char tecla = 'X';
```

Bool

Representa um valor verdadeiro ou falso.

```
bool clienteEspecial = true;
```

Byte

Pode ser um valor entre 0 e 255.

```
byte idade = 16;
```

Short

Inteiro de 16 bits com sinal. Possui valores entre -32.768 e 32.767.

```
short numero = 512;
```

Integer

Um inteiro possui 32 bits contendo valores entre: -2.147.483.648 e 2.147.483.647.

```
int posicao = 2850;
```

Tipo ponto flutuante (float)

Dois tipos de dados pertencem a esta categoria: double (precisão de 15 a 16 dígitos) e float (precisão de 7 dígitos).

```
float valor = 3.456F;  
double variancia = 1122.34556;  
double PI = 3.141521D
```

Decimal

Este tipo é um formato muito exato, visto que a precisão é de 28 casas decimais, o que o torna ideal para variáveis que armazenam cálculos financeiros, ele vem para substituir o formato Currency nas operações com moedas.

```
decimal salario = 415.35M;
```

Devemos colocar o M logo após o valor para que não seja tratada como tipo Double.

String

O tipo string costuma ser o mais utilizado em programação, onde podemos armazenar todo o tipo de conteúdo: letras, números e caracteres especiais.

```
string escola = "CTI – UNESP";
```

Se você precisar concatenar uma string, utilizamos o sinal de mais (+):

```
string descricao = "Turma de Informática " + "CTI – 2013";
```

Outro recurso importante é a facilidade de extrair um caractere da string:

```
string turma = "72C";
```

```
char extrai = turma[2]; // retorna "C" – Lembrando que em um vetor o índice inicial é 0 (zero).
```

Podemos ainda utilizar vários caracteres de escape que podem ser muito úteis:

Seqüência	Significado	Seqüência	Significado
\'	Apóstrofo	\f	Avanço de Página
\"	Aspas	\n	Salto de Linha
\\	Barra Invertida	\r	Retorno de Carro
\a	Alerta	\t	Tabulação Horizontal
\b	Retrocesso	\v	Tabulação Vertical

Enum

É utilizado quando queremos criar um tipo distinto, constituído de um conjunto de constantes nomeadas.

```
enum Dias {Domingo, Segunda, Terça, Quarta, Quinta, Sexta, Sábado};
```

Os elementos da enumeração são do tipo int. Os outros tipos que podemos utilizar são long, short e byte. Podemos acessar o valor associado a cada elemento da enumeração:

```
int x = (int) Dias.Domingo; // retorna para x o valor 0
```

```
int y = (int) Dias.Sexta; // retorna para y o valor 5
```

Podemos também atribuir valores arbitrários a cada elemento:

```
enum Meses {Jan=100, Fev=200, Mar=300, Abr=400, Mai=500, Jun=600, Jul=700};
```

Array

Array é uma coleção de elementos armazenados em sequência. No C#, o primeiro elemento de um array contém o índice zero (0). Um array geralmente é caracterizado pelo número de dimensões que possui. Um array pode conter uma única dimensão ou ser multidimensional.

Array de dimensão única

```
int[] qtdeMes = new int[12];
```

```
int[] idade = new int[5] {16,18,22,24,15};
```

```
string[] cidade = new string[3] {"Bauru", "Marília", "Jaú"};
```

Podemos também omitir o número de elementos:

```
int[] valor = new int[] {10,20,30,40,50};
```

```
string[] turma = new string[] {"72A", "72B", "72C", "72D", "73C"};
```

```
string[] dias = {"Seg", "Ter", "Qua", "Qui", "Sex", "Sab", "Dom"};
```

Array multidimensional

```
int[,] posPecas = new int[3,2] { {5,2}, {2,3}, {3,4} };
```

```
string[,] tempo = new string[2,3] { {"Jan", "Temp", "Vlr Temp"}, {"Fev", "Umid.", "Vlr Umid."} };
```

```
char[,] forca = new char[3,3] { {'X','O','O'}, {'O','X','O'}, {'X','O','X'} };
```

Operadores

Os operadores são muito utilizados quando precisamos efetuar uma operação matemática, condicional, etc. Segue abaixo os operadores suportados pelo C#:

Categoria	Operadores
Primários	(x) a[i] x++ y--
Unários	+ - ! ++x --x soma++
Aritméticos	+ - * / %
Shift	>> <<
Relacional	<> <= >= is
Igualdade	== !=
E lógico	&
Ou exclusivo lógico	^
Ou lógico	
E condicional	&&
Ou condicional	
Condicional	?:
Atribuição	= += -= *= /= %= <<= >>= &= ^= =

14. ESTRUTURAS DE CONTROLE

As estruturas de controle utilizadas no C# são semelhantes às que encontramos em outras linguagens como o JScript e o C++, com pequenos detalhes, como por exemplo a instrução switch ao invés de Select Case (Visual Basic).

Instrução if

O if é a instrução mais utilizada e encontrada em praticamente todas as linguagens de programação. Você pode aninhar várias condições para testar diferentes níveis dentro de uma aplicação.

```
if (dias == 0) {
    valor = 0;
}
else if dias > 100) {
    valor = 100;
}
else {
    valor = 500;
}
```

Instrução switch

A instrução switch geralmente é utilizada quando precisamos testar uma grande quantidade de condições para uma variável ou expressão.

```
int mes = Convert.ToInt32(txtMes.Text);
double percentual=0;
switch (mes)
{
    case 1:
        percentual = 2.15;
        break;
    case 2:
        percentual = 3.11;
        break;
    case 3:
        percentual = 4.19;
        break;
    default:
        percentual = 8.90;
        break;
}

MessageBox.Show(Convert.ToString(percentual));
```

Técnico em Informática

Utilizando loopings

Os loopings (laços/repetição) são muito úteis para percorrer uma grande quantidade de registros ou itens em um array, possibilitando o tratamento de cada item ou registro de modo especial.

```
int i = 0;
int soma = 0;
while (i <= 10)
{
    soma += i;
    i += 2;
}
Console.WriteLine("Soma dos pares até 10: " + Convert.ToString(soma));
```

While

A instrução **while** permite a repetição de uma ou um bloco de instruções, este processo de repetição ocorrerá enquanto a condição que é testada logo no início seja verdadeira.

```
int resultado, numero = 5, mult = 0;
while (mult <= 10);
{
    resultado = numero * multiplicador;
    mult = mult + 1; // ou utilizando o operador de incremento: mult++;
}
```

Do ... while

A instrução **do ... while** só difere do **while** em relação ao teste da condição, que é realizado no final, verificando se é preciso repetir a execução do código.

```
string mensagem = "";
int valor = 10;
do
{
    mensagem = mensagem + Convert.ToString(valor) + " - ";
    valor--;
}
while (valor != 0);
```

For

A instrução **for** é utilizada para repetir uma instruções ou um bloco de instruções, até que a condição seja verdadeira. Ela é mais utilizada quando sabemos a quantidade de vezes que desejamos repetir o bloco de comandos.

A instrução for tem 3 parâmetros:

```
for (inicializador;    condição;    repetidor)

int somaPar = 0;
for (int num = 0; num <= 20; num++)
{
    if (num % 2 == 0)
    {
        somaPar = somaPar + num;
    }
}
```

Foreach

Utilizamos a instrução foreach para enumerar itens de um array ou coleção.

A sintaxe é:

foreach (TipodeDados NomedaVariável in NomeArray/NomeColeção)

```
int[] num = new int[] { 0, 1, 2, 3, 4, 5, 6 };
string linha = "";
foreach (int elem in num)
{
    Linha+= Convert.ToString(elem) + " - ";
}
```

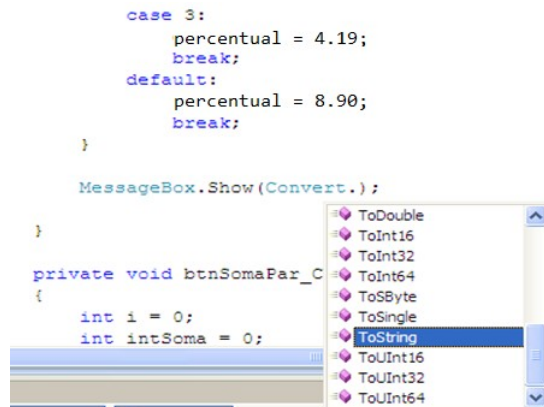
}

15. Métodos para Conversão de Tipos de Dados

A linguagem C# é fortemente tipada, com isto é necessário a conversão correta de acordo com o tipo de dados que será necessário armazenar. A classe System.Convert possibilita a realização das conversões.

```
int mes = Convert.ToInt32(txtMes.Text); // Aqui estamos convertendo o conteúdo da
caixa de texto txtMes para uma variável do tipo valor inteiro de 32 bits.
```

```
MessageBox.Show(Convert.ToString(percentual)); // Neste caso temos a conversão do
valor de precisão dupla da variável para um tipo string para ser utilizado no
MessageBox.Show.
```





Unidade IV

Carga Horária Prevista

- Três aulas

Objetivos

- Conceituar a modularização de programas;
- Utilizar métodos e classes;
- Utilizar os métodos da classe Math – Random - Directory;
- Utilizar as caixas de diálogo padrão do Windows.

Conteúdos

- Métodos em C#
- Criação de métodos pelo programador
- Classes Math – Random – Directory
- Carregamento de imagens dinâmicas (em tempo de execução)

16. Métodos

A maioria dos programas que resolvem problemas reais é muito maior que os nossos exemplos e exercícios de sala de aula. A experiência tem mostrado que a melhor maneira de desenvolver e manter um programa grande é construí-lo a partir de pequenas partes simples, ou módulos. Essa técnica é conhecida como dividir e conquistar.

17. Módulos de programa em C#

Em C#, os módulos são chamados de métodos e classes.

Os programas são escritos por meio da combinação de novos métodos e classes que o programador escreve, com métodos e classes “previamente empacotados”, disponíveis no .Net Framework Class Library (FCL).

O FCL fornece uma rica coleção de classes e métodos para efetuar cálculos matemáticos, manipulação de strings, operações de entrada/saída, verificação de erro e muitas outras, como exemplo, as classes Console e MessageBox.Show() usadas em exemplos anteriores.

O programador pode escrever métodos para definir tarefas específicas, que podem ser usados em muitos pontos em um programa (métodos definidos pelo programador), permitindo a modularização do programa.

Um método é chamado (isto é, solicitado a executar uma tarefa designada) por uma chamada a método, que especifica o nome do método e pode fornecer informações (como argumentos) que o método chamado exige para executar sua tarefa. Ao término da execução das tarefas do método ele retorna um resultado para o método chamador. (Function)

As variáveis declaradas nas definições de método são variáveis locais – apenas o método que as exibe sabe da sua existência. A maioria dos métodos tem uma lista de parâmetros que permite que as chamadas ao método comuniquem informações entre os métodos. Esses parâmetros também são variáveis locais para esse método.

```
public float VelocMedia(float distanciaKM, float tempoMin) // Parâmetros = variáveis locais
{
    float VelocidadeMedia; // Variável Local do Método VelocMedia
    if (tempoMin==0)
        return 0;
    else
    {
        VelocidadeMedia = distanciaKM / (tempoMin / 60);
        return velocidadeMedia;
    }
}
```

18. Métodos da classe Math

Os métodos da classe Math permitem que o programador efetue certos cálculos matemáticos comuns, como exemplo para calcular e imprimir a raiz quadrada de 900 (o método Math.Sqrt, recebe o argumento um double e retorna um resultado do tipo double), poderia escrever:

```
Console.WriteLine(Math.Sqrt(900.0));
```

19. Geração de números aleatórios

Em diversos aplicativos, principalmente de simulação e jogos, temos o elemento da chance, da sorte, que pode ser fornecido através de números aleatórios, no C# através da classe Random (localizada na namespace System) podemos implementar essas funcionalidades.

```
Random sorteio = new Random(); // crio / inicializa um objeto do tipo Random;
int numSorteio = sorteio.Next(); // método Next()
```

O método Next() gera um valor aleatório, do tipo int, positivo entre zero e a constante Int32.MaxValue (o valor 2.147.483.647).

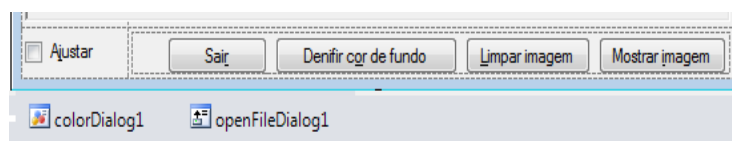
Podemos definir argumentos no método Next():

```
int moeda = sorteio.Next(2); // Retorna número 0 e 1
int sorteioDado = sorteio.Next(1,7); // Retorna números no intervalo de 1 a 6
```

Exemplo: Vamos gerar 10 cartões de jogos para a Mega Sena.

20. Carregando imagens dinamicamente

Objetos que permitem a apresentação de imagens possuem a propriedade Image, onde em tempo de desenvolvimento, podemos carregar uma imagem fixa, como exemplo um logo da empresa que será apresentado no formulário.



Podemos utilizar os componentes de caixa de diálogo para fazer o carregamento de imagens ou ainda ajustar cores, dentre outros:

```
private void btnMostrarImagem_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK )
    {
        picImagem.Load(openFileDialog1.FileName);
    }
}

private void btnDefinirCor_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK )
    {
        picImagem.BackColor = colorDialog1.Color;
        this.BackColor = colorDialog1.Color;
    }
}

private void btnLimparImagem_Click(object sender, EventArgs e)
{
    picImagem.Image = null;
}

private void btnSair_Click(object sender, EventArgs e)
```

```
{ this.Close(); }
```

```
private void chkAjustar_CheckedChanged(object sender, EventArgs e)
{
    if (chkStretch.Checked)
        picImagem.SizeMode = PictureBoxSizeMode.StretchImage;
    else
        picImagem.SizeMode = PictureBoxSizeMode.Normal;
}
```

Normalmente existe a necessidade do carregamento de uma imagem em tempo de execução, de acordo com algum evento ou uma condição deve ser carregado um tipo de imagem, para tanto temos o método `Image.FromFile`, onde através do método `Directory.GetCurrentDirectory` pode obter o caminho da pasta em que o programa está sendo executado.

```
using System.IO;
```

```
public void mostraDado(Label dado) // ou PictureBox
{
    int face = sorteio.Next(1,7);

    imgDado.Image = Image.FromFile(Directory.GetCurrentDirectory() + "\\imagens\\
dado" + face + ".gif" )
}
```

Exemplo: Apresentar o sorteio aleatório de 4 dados.

Acesso a Banco de Dados com C#

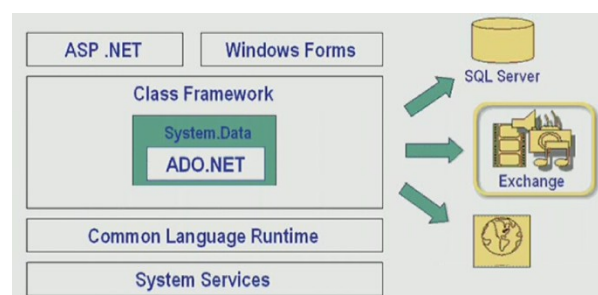
Normalmente os softwares que escrevemos necessitam de um acesso a dados, como exemplo uma aplicação web onde as informações apresentadas são lidas de um banco de dados (catálogo de produtos, compras realizadas), ou ainda quando enviamos informações para serem gravadas no banco de dados (como dados do cliente, compra de um produto).

Os gerenciadores de banco de dados possibilitam a manipulação dessas informações, um exemplo típico de um banco de dados para gerenciar as vendas de uma loja, incluem lista de clientes, cadastro de produtos e fornecedores, lista e controle das vendas realizadas, gerar comissões e folha de pagamento dos funcionários, sendo que temos a disposição os bancos de dados relacionais: SQL Server e Oracle, PostgreSQL, MySQL e o Microsoft Access.

Visão Geral do ADO .NET

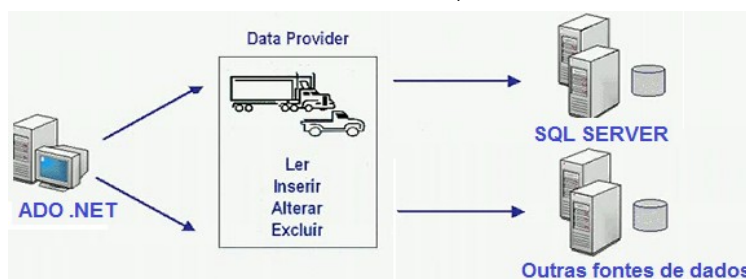
O ActiveX® Data Objects para .Net Framework (ADO.NET) disponibiliza muitas inovações para o acesso a dados, incluindo a possibilidade de gerenciamento em um ambiente desconectado.

O ADO .NET é um conjunto de classes as quais permitem que as aplicações baseadas em .Net leiam e atualizem informações em banco de dados e em outras fontes de dados.



O acesso a estas fontes de dados é realizado através da utilização de provedores de dados do .Net, sendo que basicamente temos os provedores de dados específicos (nativos) para determinado banco de dados, como o SQL Server Data Provider (ou mesmo Oracle e e PostgreSQL).

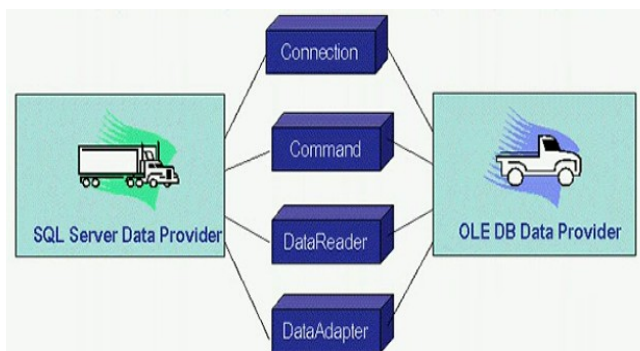
Podemos utilizar ainda, os provedores OLEDB que permitem a conexão a vários tipos de bancos de dados que dispõem de suporte para esta biblioteca.



Como estes provedores permitem acesso a tipos específicos de fontes de dados, utilizamos o namespace:

- System.Data.SqlClient para acessarmos o banco SQL Server, de forma nativa.
- System.Data.OleDb para acessarmos qualquer outra fonte de dados suportada pelo OLEDB.

Cada um destes provedores possuem quatro objetos principais:



- Connection: responsável pela conexão com a fonte de dados;
- Command: para executar comandos de manipulação SQL e opcionalmente retornar dados de uma fonte de dados;
- DataReader: processa um comando para leitura de dados;
- DataAdapter: para trocar dados entre a fonte de dados e um DataSet.

Objeto Connection

Como você já deve ter desconfiado, o objeto **Connection** tem a função de gerar uma conexão com uma fonte de dados sendo, portanto o objeto fundamental no acesso a dados.

Para estabelecer uma conexão com uma fonte de dados o objeto Connection utilizamos a propriedade **ConnectionString** que é uma string de conexão que deverá ser informada para que a conexão seja efetivamente aberta.

Para conectarmos a um banco de dados devemos:

- Configurar o tipo de conexão.
- Especificar a fonte de dados.
- Conectar a fonte de dados.
- Quando encerrarmos o trabalho com os dados, fechamos a conexão.

SQL Server utilizando SqlConnection

```
using System.Data;
```

// Acesso nativo

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

```
SqlConnection cn = new SqlConnection();  
cn.ConnectionString = "Data Source=localhost;" +  
                    "Integrated Security=SSPI;" +  
                    "Initial Catalog=Northwind";
```

```
static string stringConexao = "Data Source = LDI310\\SQLEXPRESS; " +  
                              "Initial Catalog = agenda;" +  
                              "User ID = usercti; Password = cti123";  
SqlConnection cn = new SqlConnection();
```

```
cn.ConnectionString = stringConexao;  
cn.Open();
```

SQL Server utilizando OleDbConnection

```
using System.Data;
```

```
using System.Data.OleDb;
```

```
static string stringConexao = "Provider=SQLOLEDB; " +  
                              "Data Source = LDI310\\SQLEXPRESS; " +  
                              "Initial Catalog = agenda; " +  
                              "User ID = userCTI; Password = cti123";
```

```
OleDbConnection cn = new OleDbConnection();
```

```
cn.ConnectionString = stringConexao;  
cn.Open();
```

Postgres utilizando OLEDBConnection

```
using System.Data;
```

```
using System.Data.OleDb;
```

```
string stringConexao = "Provider= PostgreSQL; " +  
                      "Data Source = LDI210; " +  
                      "Location = painelao; " +  
                      "User ID = alunocti; Password = 123";
```

```
OleDbConnection cn = new OleDbConnection();
```

```
cn.ConnectionString = stringConexao;  
cn.Open();
```

Objeto Command

Os objetos Command são usados para executar declarações SQL e procedimentos armazenados (stored procedures). Os métodos usados para realizar estas tarefas são:

- **ExecuteNonQuery** - executa declarações SQL que não retornam dados, tais como **INSERT, UPDATE, DELETE e SET**.
- **ExecuteScalar** - retorna um valor único como resultado de uma função agregada: **SUM, AVG, COUNT, MAX E MIN**.
- **ExecuteReader** - executa declarações SQL que retornam linhas de dados, tais como o **SELECT**.
- **ExecuteXMLReader** - retorna um objeto XMLReader com a leitura de um arquivo XML.

Para criar um comando você já deve ter uma conexão criada. Assim para um banco de dados SQL Server devemos usar um objeto **SqlCommand**, já se usarmos provedores OLE DB deveremos usar o objeto **OleDbCommand**. Vejamos um exemplo de criação de um comando:

```
// variável para armazenar o comando SQL
string stringSQL = "INSERT INTO Pessoa (Nome, Sobrenome, Idade, " +
                  "Altura, DataNascimento) " +
                  "VALUES ('" + txtNome.text + "', " +
                  "'" + txtSobrenome.text + "', " +
                  txtIdade.text + ", " +
                  txtAltura.text + ", " +
                  "'" + txtDataNascimento.text + "')";

// cria um objeto do tipo Command vinculado a conexão
SqlCommand cmd = new SqlCommand(stringSQL, conn);

// ExecuteNonQuery é utilizado para enviar um comando SQL de atualização
// Tipo Insert - Update - Delete
cmd.ExecuteNonQuery();
```

Criando Commands mais robustos

O exemplo que vimos anteriormente cria a string SQL dinamicamente, isto é fácil para o entendimento e rápido para programas, mas apresenta sérios problemas:

- Os usuários podem acidentalmente entrar com caracteres que afetarão os seus comandos SQL, por exemplo um nome que contém apóstrofo, ao ser adicionado na string não será válido: 'Andre's' '150',12
- Usuários expertos podem realizar o que conhecemos como Ataque de SQL Injection, fazendo com que comandos SQL tenham comportamento violado, permitindo inclusive que um usuário faça o login em um sistema sem informar usuário e senha.

Uma forma mais robusta e conveniente é utilizarmos Command parametrizado, onde as informações na string SQL, serão substituídas pelos parâmetros. Exemplo:

```
string stringSQL = "select * from operadora " +
    "where id_operadora = " + txtId_Operadora.Text;

string stringSQL = "select * from operadora " +
    "where id_operadora = @Id_Operadora";
```

Para adicionarmos o conteúdo ao parâmetro:

```
cmd.Parameters.AddWithValue("@Id_Operadora", txtId_Operadora.Text);
```

Segue agora um exemplo com o comando Insert:

```
// variável para armazenar strings de comandos SQL
string stringSQL = "insert into pessoa (nome, sobrenome, idade,
    altura, datanascimento) " +
    "VALUES (@nome, @sobrenome, @idade,
    @altura, @datanascimento)";

// cria um objeto do tipo Command vinculado a conexão
SqlCommand cmd = new SqlCommand(stringSQL, conn);

cmd.Parameters.AddWithValue("@nome", txtNome.Text);
cmd.Parameters.AddWithValue("@sobrenome", txtSobrenome.Text);
cmd.Parameters.AddWithValue("@idade", Int32.Parse(txtIdade.Text));
cmd.Parameters.AddWithValue("@altura", float.Parse(txtAltura.Text));
cmd.Parameters.AddWithValue("@datanascimento",
    DateTime.Parse(txtDataNascimento.Text));

// ExecuteNonQuery é utilizado para enviar um comando SQL de
// atualização (Insert - Update - Delete)
cmd.ExecuteNonQuery();
```

Temos uma pequena variação para utilizarmos parâmetros através de Providers OLEDB, onde devemos utilizar o caracter ? para indicar que nesta posição será carregado um parâmetro:

```
string stringSQL = "insert into operadora (nome, prefixo) " +
    " values (?, ?) ";

OleDbCommand cmd = new OleDbCommand(stringSQL, cn);

cmd.Parameters.AddWithValue("@nome", txtNome.Text);
cmd.Parameters.AddWithValue("@prefixo", txtPrefixo.Text);
cmd.ExecuteNonQuery();
```

Objeto DataReader

O DataReader é um objeto de leitura de dados, rápido e eficiente. Ele promove a leitura “somente adiante”, sem possibilidade de navegar registros para trás, e também é read-only, ou seja não permite alteração dos dados.

A grande vantagem do DataReader é a sua velocidade: ele deve ser usado quando for necessário ler grandes quantidades de dados que não serão modificados como, por exemplo, em relatórios ou listagens, ou ainda para o carregamento de dados na tela.

O DataReader é específico de cada provedor: SqlDataReader, OracleDataReader, OleDbDataReader, OdbcDataReader.

O DataReader é uma classe estática e nunca deve ser instanciado. Ele deve sempre receber a referência de outro objeto que retorne um DataReader, como por exemplo um objeto Command.

Segue um exemplo para carregar os dados na tela:

```
private void txtId_Operadora_Validated(object sender, EventArgs e)
{
    string stringSQL = "select * from operadora " +
                      "where id_operadora = ?";

    OleDbCommand cmd = new OleDbCommand(stringSQL, cn);
    cmd.Parameters.AddWithValue("@id_operadora", txtId_Operadora.Text);

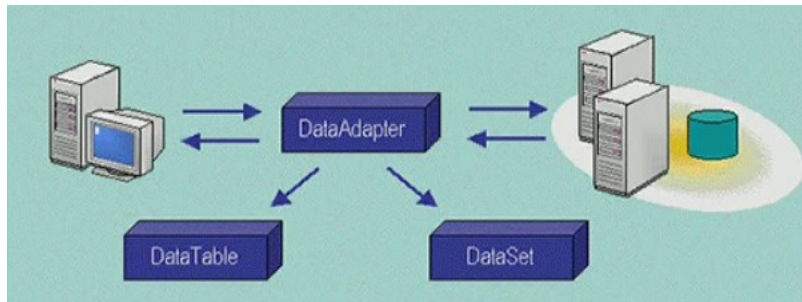
    OleDbDataReader dr = cmd.ExecuteReader();

    if (dr.Read())
    {
        txtNome.Text = (string) dr["nome"];
        txtPrefixo.Text = (string) dr["prefixo"];

        dr.Close();
        dr.Dispose();
    }
    else
    {
        limpaForm();
        MessageBox.Show("Operadora não encontrada !!!", "Aviso",
                        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
```

Objeto DataAdapter

Utilizamos o DataAdapter para trocar dados entre uma fonte de dados e um DataSet, recuperar dados apropriados e inseri-los nos objetos DataTable, e para atualizar estas informações do DataSet na fonte de dados.



Segue um exemplo utilizando a conexão existente e um objeto command para instanciar um DataAdapter, preencher o DataSet e carrega-lo num DataGrid:

```
private void carregaGrid()
{
    string stringSQL = "select * from pessoa order by nome";
    SqlCommand cmd = new SqlCommand(stringSQL, conn);

    SqlDataAdapter daPessoa = new SqlDataAdapter(cmd);
    DataSet dsPessoa = new DataSet();

    // É preciso esvaziar o repositório de dados antes do método
    // Fill senão ocorre a duplicação de dados
    dsPessoa.Clear();

    // O método Fill lê os dados conforme o comando SQL e armazena
    // no DataSet que é um repositório de dados
    daPessoa.Fill(dsPessoa);

    // Atribui o DataSet Pessoa ao Grid dgPessoas
    dgPessoas.DataSource = dsPessoa.Tables[0];
}
```

Podemos utilizar um DataTable para carregar um ComboBox apresentando os nomes das pessoas:

```
// Utilizando Data Table
DataTable dtPessoa = new DataTable();

string stringSQL = "select id_pessoa, nome from pessoa order by nome";
SqlCommand cmd = new SqlCommand(stringSQL, conn);

SqlDataAdapter daPessoa = new SqlDataAdapter(cmd);

dtPessoa.Clear();
daPessoa.Fill(dtPessoa);

cmbPessoa.DataSource = dtPessoa;
cmbPessoa.ValueMember = "id_pessoa";
cmbPessoa.DisplayMember = "nome";
cmbPessoa.SelectedIndex = 0;
```

Tipos de Dados PostgreSQL/Npgsql x .Net

Npgsql suporta os seguintes tipos de dados:

Postgresql Type	NpgsqlDbType	System.DbType Enum	.Net System Type
int8 -9223372036854775808 a 9223372036854775807	Bigint	Int64	Int64
Bool	Boolean	Boolean	Boolean
Box, Circle, Line, LSeg, Path, Point, Polygon	Box, Circle, Line, LSeg, Path, Point, Polygon	Object	Object
Bytea	Bytea	Binary	Byte[]
Date	Date	Date	DateTime, NpgsqlDate
float8 (precisão 15 casas)	Double	Double	Double
int4 -2147483648 a 2147483647	Integer	Int32	Int32
money (sem limitação)	Money	Decimal	Decimal
numeric (sem limitação)	Numeric	Decimal	Decimal
float4 (precisão 15 casas)	Real	Single	Single
int2 -32768 a 32767	Smallint	Int16	Int16
Text	Text	String	String

Postgresql Type	NpgsqlDbType	System.DbType Enum	.Net System Type
varchar	Varchar	String	String
Time	Time	Time	DateTime, NpgsqlTime
Timetz	Time	Time	DateTime, NpgsqlTimeTZ
Timestamp	Timestamp	DateTime	DateTime, NpgsqlTimestamp
Timestamptz	TimestampTZ	DateTime	DateTime, NpgsqlTimestampTZ
Interval	Interval	Object	TimeSpan, NpgsqlInterval
Inet	Inet	Object	NpgsqlInet, IPAddress (there is an implicit cast operator to convert NpgsqlInet objects into IPAddress if you need to use IPAddress and have only NpgsqlInet)
Bit	Bit	Boolean	Boolean, Int32 (If you use an Int32 value, odd values will be translated to bit 1 and even values to bit 0)
Uuid	Uuid	Guid	Guid
Array	Array	Object	Array In order to explicitly use array type, specify NpgsqlDbType as an 'OR'ed type: NpgsqlDbType.Array NpgsqlDbType.Integer for an array of Int32 for example.
bigserial 1 a 9223372036854775807	Bigint	Int64	Int64
serial 1 a 2147483647	Integer	Int32	Int32

Tipos de dados Serial / BigSerial

Técnico em Informática

Os tipos de dados serial e bigserial não são tipos verdadeiros, mas meramente uma conveniência para criar uma coluna de identificador único (similar a propriedade AUTO_INCREMENT suportada por outros bancos de dados. A criação da tabela abaixo:

```
CREATE TABLE fabricantes (  
    id bigserial NOT NULL,  
    nome character varying(50) NOT NULL,  
    CONSTRAINT pk_fabricantes PRIMARY KEY (id) );
```

É equivalente a:

```
CREATE SEQUENCE fabricantes_id_seq  
INCREMENT 1  
MINVALUE 1  
MAXVALUE 9223372036854775807  
START 1  
CACHE 1;
```

```
CREATE TABLE fabricantes (  
    id bigint NOT NULL  
        DEFAULT nextval ('fabricantes_id_seq'),  
    nome character varying(50) NOT NULL,  
    CONSTRAINT pk_fabricantes PRIMARY KEY (id) );
```