REVISTA CONEXÃO ELETRÔNICA

MANIPULAÇÃO DE ARQUIVOS NA LINGUAGEM C#: StreamWriter e StreamReader

Frederico Augusto Dejavite
Graduando em Engenharia da Computação
Faculdades Integradas de Três Lagoas – FITL/AEMS

Lucas Nuud Táparo
Graduando em Engenharia da Computação
Faculdades Integradas de Três Lagoas – FITL/AEMS

Victor Rafael Terto Lopes
Graduando em Engenharia da Computação
Faculdades Integradas de Três Lagoas – FITL/AEMS

Alan Pinheiro de Souza
Mestre em Sistemas de Informação
Docente das Faculdades Integradas de Três Lagoas – FITL/AEMS

Thiago de Oliveira Correia
Especialista em Desenvolvimento de Jogos Digitais
Docentes das Faculdades Integradas de Três Lagoas – FITL/AEMS

RESUMO
Este artigo tem como objetivo apresentar uma visão geral da linguagem C#, destacando suas características e apresentando alguns ambientes de desenvolvimento. O foco do projeto é discutir a leitura e escrita de arquivos no sistema operacional *Microsoft Windows*. Para isso foi realizada uma pesquisa bibliográfica sobre a linguagem, além disso, utilizou-se o paradigma de orientação a objetos na elaboração de códigos visando exemplificar a utilização das funcionalidades de leitura e escrita de arquivos. Todos os exemplos foram desenvolvidos no ambiente de programação *Visual Studio*, que foi a principal ferramenta adotada ao longo do projeto. Os resultados alcançados demonstram a facilidade na aplicação das funcionalidades *StreamWriter* e *StreamReader* e na organização do projeto dentro do ambiente de desenvolvimento.

PALAVRAS-CHAVE: *C#; Sharp; StreamWriter; StreamReader;* Visual Studio; .NET.

INTRODUÇÃO

A linguagem C# foi criada por *Anders Hejlsberg* em meados de 2000, hoje é uma linguagem muito usada e totalmente direcionada à plataforma .NET da *Microsoft*. É uma linguagem conhecida por sua facilidade na construção de estruturas de programação, bem como flexibilidade e sendo totalmente orientada a objetos. O intuito da criação do C# foi ser totalmente desenhado à medida do .NET

com vários objetivos: robustez, orientação a objetos, preservação de investimentos, entre outros. Segundo (STELLMAN; GREENE, 2008), “O C# é uma linguagem de programação poderosa e uma ferramenta valiosa na palma da sua mão”. Contando com estes benefícios citados, ajudará bastante na organização e desenvolvimento das estruturas usadas nas demonstrações.

Além da seção inicial de introdução e das seções de encerramento que apresentam as considerações finais da pesquisa e as referências bibliográficas adotadas no trabalho, esse artigo está dividido em três seções. A primeira discute as características da linguagem e benefícios. A segunda que apresenta dois ambientes de desenvolvimento mais usados no mercado. A terceira apresenta as implementações do tema proposto, a leitura e escrita de arquivos com exemplos práticos.

1 CARACTERÍSTICAS DO C#

Ao longo desta seção serão demonstradas três características que são levadas em conta quando se fala em qual linguagem de programação escolher para desenvolver um novo sistema. Como com quais tipos de linguagem os sistemas serão integrados, gerenciamento de memória dos computadores usados e investimentos exigidos no projeto. São várias dúvidas que podem ser sanadas com as características como: robustez, orientação a objetos e preservação de investimentos.

1.1 Robustez

Segundo Manzano (2012), pode-se considerar como robustez todas as características que trazem segurança e conforto para o desenvolvimento na linguagem. Gestão automática de memória é um bom exemplo, nela o GC (*Garbage Collector*) é responsável por "desalocar" quantidades de memória de um programa que já não se encontram mais acessíveis. É um recurso que uma vez acionado, é totalmente automatizado, sem a necessidade de desenvolvimento de comandos para alocação de memória. Também se pode citar um melhor tratamento de exceções (tratamento de erros). Eles estão acessíveis para todas as

bibliotecas.NET. Outra qualidade é a interoperabilidade, ou seja, sua capacidade de comunicação com várias outras linguagens como: XML[1], SOAP[2], COM[3] e qualquer linguagem do .NET *Framework* (STIEFEL; OBERG, 2002).

1.2 Orientação a Objetos

Segundo (STIEFEL; OBERG, 2002), a orientação a objetos é um conceito muito utilizado nos dias de hoje. Um conceito que está relacionado com a idéia de classificar, organizar e abstrair coisas. Uso de classes, herança, polimorfismo e encapsulamento, segundo Deitel *et al.* (2003), são recursos de desenvolvimento que tornam o C# uma linguagem versátil e poderosa.

1.3 Preservação de Investimentos

Segundo Deitel *et al. (*2003), existem várias características no C# que facilitam o trabalho dos programadores. Essa linguagem disponibiliza um conjunto abrangente de recursos para empresas, e assim se pode dizer que há uma preservação de investimentos. Esta preservação é devida à fácil integração com outras linguagens, melhor produtividade, fácil criação de objetos, semelhanças com C++ e Java. Outra característica importante para Lima (2002) é a curva de aprendizagem pequena, isso facilita muito no treinamento de novos profissionais no mercado de trabalho onde a rotatividade está em alta devido a grande procura de profissionais da área.

2 AMBIENTES DE DESENVOLVIMENTO

Os ambientes de desenvolvimento, também chamado de IDE (*Integrated Development Enviroment*), no português Ambiente Integrado de Desenvolvimento, são programas que ajudam a editar seus códigos, gerenciar seus arquivos e publicar seus projetos. Para Deitel *et al.* (2003, p.53), com um IDE, um programador pode criar, executar, testar e depurar uma fração do código, o que gastaria mais tempo sem usar o IDE. Geralmente é a melhor ferramenta do programador quando novos projetos são criados. Um ambiente integrado de desenvolvimento ajuda muito no

desenvolvimento de qualquer tipo de aplicação de *software*, seja ele pequeno ou de grande complexidade.

Existe um ganho considerável em tempo quando se usa o IDE. A organização dos módulos e arquivos facilita na visualização geral do contexto da aplicação que está sendo criada. Quando se fala em organização no IDE, refere-se também à larga escala de ferramentas que possui e que ajuda na estruturação do projeto, criação de classes em arquivos distintos do projeto principal (*Main*), também pela diferenciação de cores nas linhas de código.

Existem vários programas IDE, a seguir serão apresentados dois dos mais usados no mercado, isto ajudará a ter uma melhor idéia de como são os ambientes de desenvolvimento do C#. A Figura 1 ilustra o *Visual Studio* 2010, uma ferramenta da *Microsoft* que é a mais usada para o C#, ela disponibiliza várias outras linguagens para desenvolvimento, tais como: ASP.NET[4], F#[5], VB.NET[6]. Este é o IDE que será usado para exemplos de leitura e escrita de arquivos, e por isso será o mais detalhado.

Na sequência, será descrito todos os elementos destacados na Figura 1:

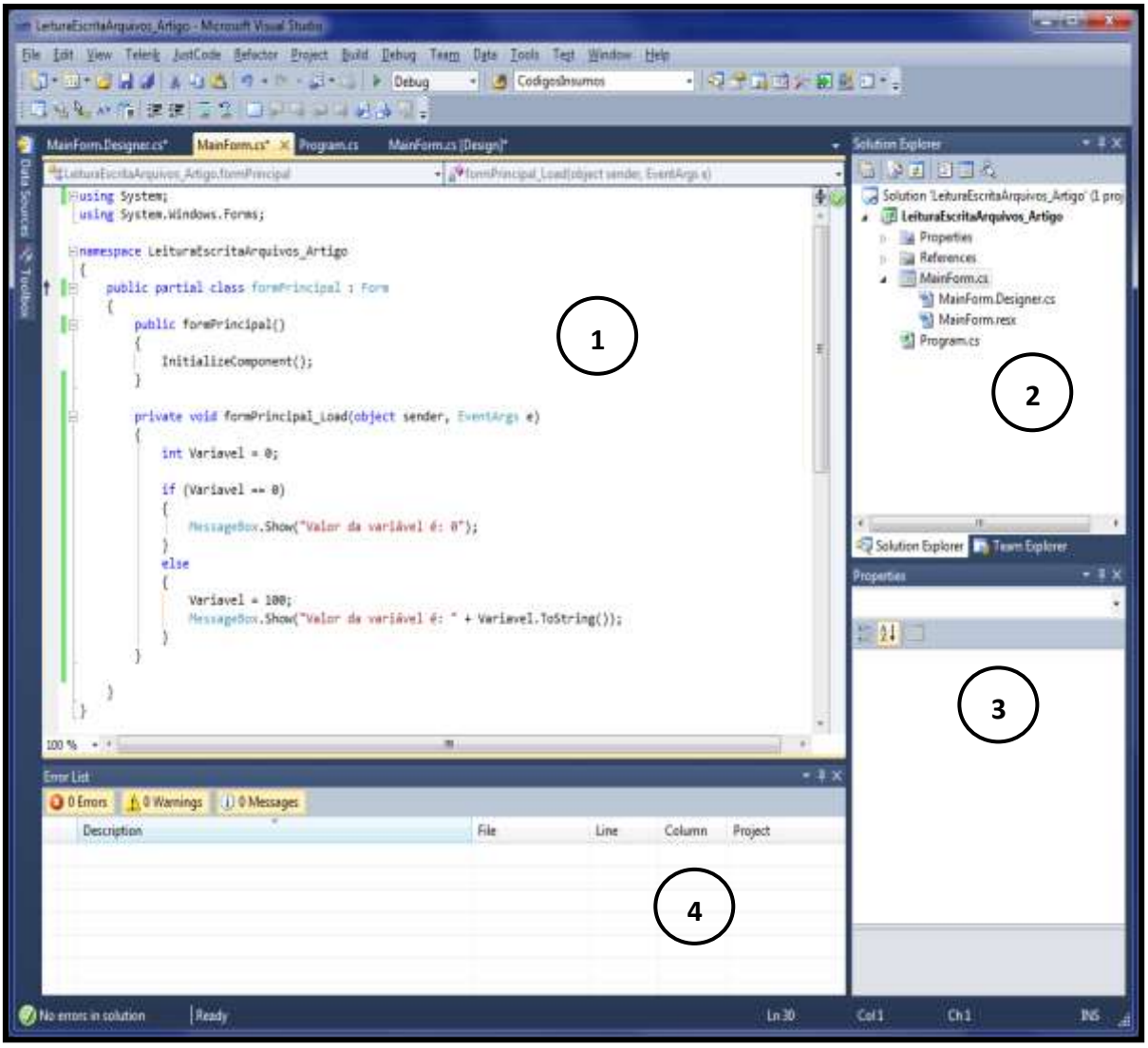
1. ***Code Editor*:** onde serão inseridos e editados todos os códigos do projeto.

2. ***Solution Explorer*:** é onde está toda a estrutura do projeto, é nele onde o desenvolvedor navega em todas as seções que ele criou.

3. **Propriedades (*Properties*):** é nela que será editada qualquer propriedade tanto de objetos criados ou do próprio projeto. Propriedades como nome do projeto, cor dor formulário, tamanho de botões, entre outras;

4. ***ErrorList:*** é a seção onde poderão ser observados todos os erros contidos no código, por exemplo, variáveis com nomes errados e conversão de tipos incompatíveis.

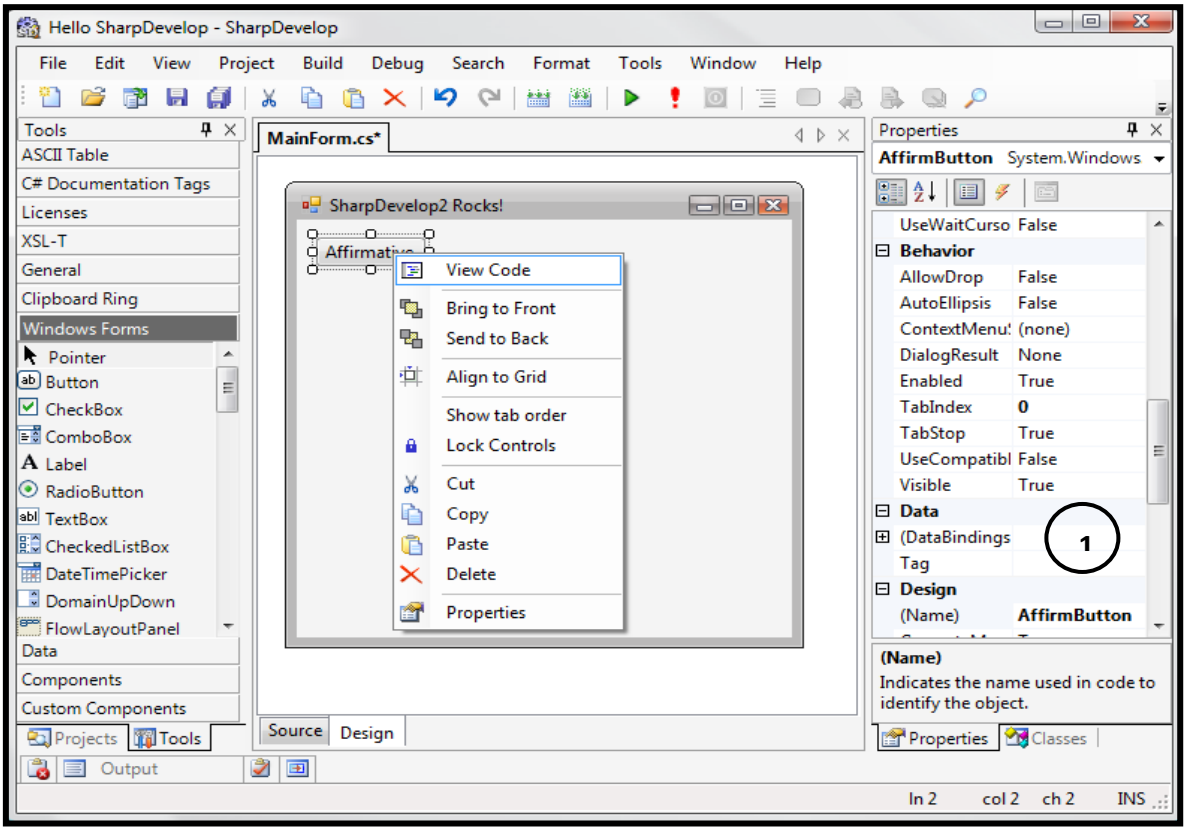
Figura 1: Visual Studio 2010.



Fonte: Elaborado pelos autores (2015).

A Figura 2 a seguir, representa o *SharpDevelop,* outro IDE que também é um dos mais utilizados por programadores da linguagem C#. É uma ferramenta gratuita da *IC#Code*, que também fornece suporte para outras linguagens. Totalmente gratuita disponibilizada no site do *IC#Code* para *download*.

Figura 2: *SharpDevelop.*



Fonte: IC#CODE (2015).

Nota-se que a estrutura do ambiente da ferramenta *SharpDevelop* é, basicamente, quase a mesma do *Visual Studio* da *Microsoft*. Observa-se na Figura 2 que um botão do formulário está sendo editado. Na sequência, será descrito o elemento destacado na Figura 2.

1. Ao lado direito está a janela *Properties* que mostra todas as propriedades do objeto selecionado, no caso o botão.

Segundo Stellman; Greene (2008), a diferenciação de cores é de grande importância, principalmente para grandes projetos porque ajuda a distinguir diferentes estruturas do código fonte e auxilia na identificação de início e fim de estruturas de códigos dentro do *Code Editor*.A Figura 3 a seguir, segue como exemplo dessa diferenciação de cores, podendo distinguir entre: variáveis, texto, classes e métodos. Pode-se observar que em uma pequena amostra de código C#, foram usadas uma variedade de recursos.

Figura 3: Código fonte C#.

```
private void formPrincipal_Load(object sender, EventArgs e)
{
    int Variavel = 0;

    if (Variavel == 0)
    {
        MessageBox.Show("Valor da variável é: 0");
    }
    else
    {
        Variavel = 100;
        MessageBox.Show("Valor da variável é: " + Variavel.ToString());
    }
}
```

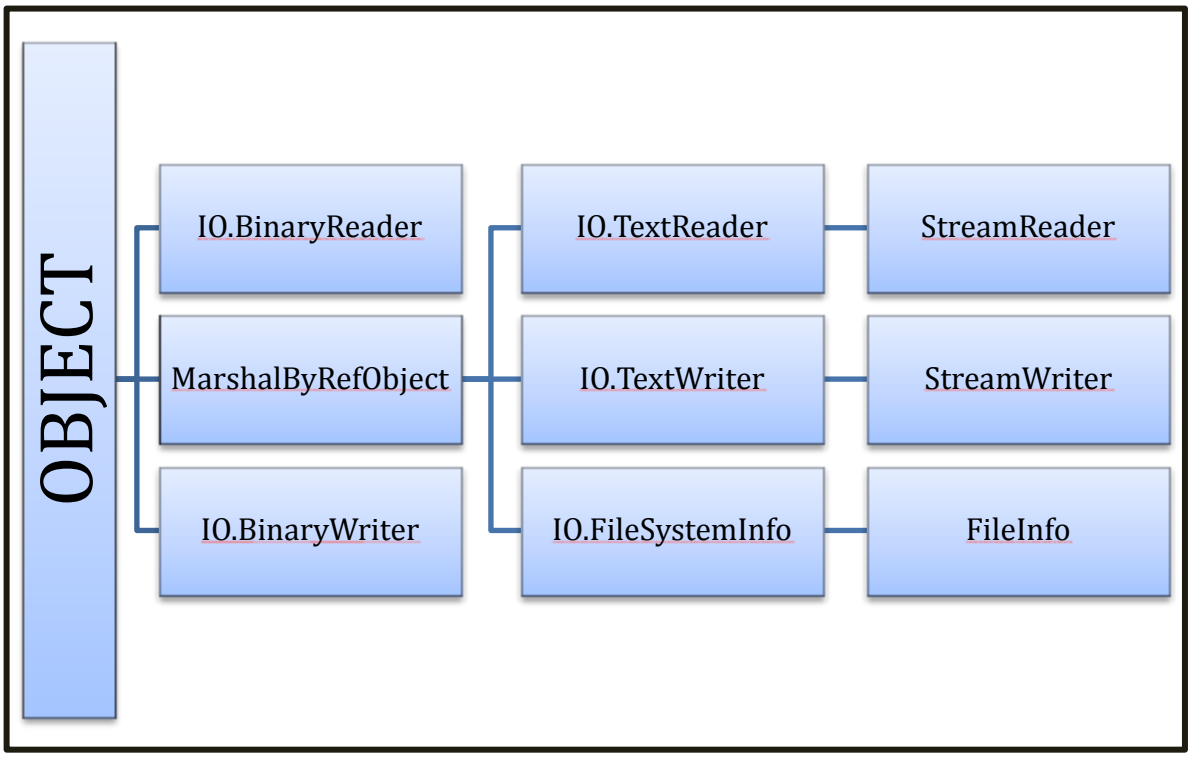
Fonte: Elaborado pelos autores (2015).

O código exibido na Figura 3 executa uma verificação toda vez que o formulário **formPrincipal** é carregado pela aplicação. Dentro deste método, ele primeiramente inicia uma variável e atribui um valor a ela, logo é executada uma verificação desta variável, onde se ela for igual à zero, uma mensagem será mostrada na tela do usuário indicando o valor contido na variável. Caso a variável seja diferente de zero, ela recebe o valor “100” e exibe outra mensagem na tela com o valor referente à variável. Observa-se que ao concatenar o texto, com o valor da variável, converte-se ela primeiramente para texto para poder ser exibido o valor junto com a mensagem. Quando é colocado **Variavel.ToString()**, uma conversão automática é feita pelo sistema.

3 MANIPULAÇÃO DE ARQUIVOS NO *WINDOWS*

Como visto anteriormente, a plataforma .NET disponibiliza vários recursos junto à linguagem C#. Serão demonstrados dois métodos que demonstram as facilidades da plataforma, nota-se que é simples quando se tem as ferramentas corretas. No caso será usado o *namespace System.IO*, que é o conjunto de classes do .NET que fornece várias funções. Nos exemplos que serão apresentados, serão usadas duas heranças contidas neste *namespace*. Na Figura 4 a seguir é demonstrada a estrutura parcial do *System.IO*.

Figura 4: Estrutura do System.IO.



Fonte: Elaborado pelos autores (2015).

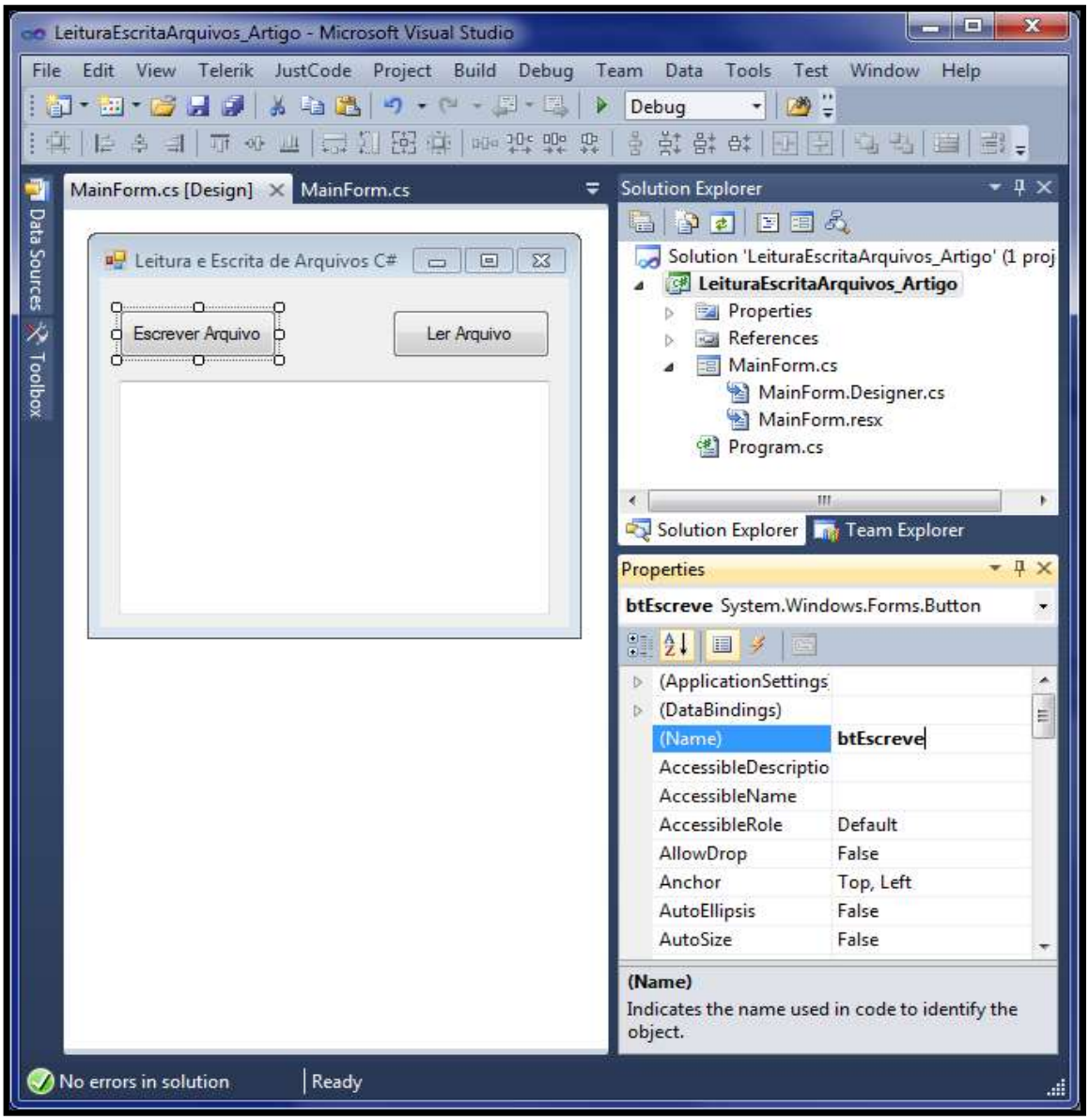
São muitas as funções fornecidas ao programador, permitindo maior facilidade no desenvolvimento de seus projetos. Todo trabalho árduo é automatizado por algumas funções, por exemplo, *FileInfo*. Quando estas são chamadas, são criadas várias instâncias em segundo plano pelo programa, como não são notadas pelo desenvolvedor, este pode direcionar a atenção para o foco principal do sistema. Duas das principais funções voltadas para a manipulação de arquivos, *StreamWriter* e *StreamReader*, serão discutidas com maiores detalhes nas próximas seções.

3.1 Escrita de Arquivos – *StreamWriter*

Segundo Microsoft (2015), o *StreamWriter* é a função que irá gravar os arquivos no disco rígido do sistema operacional. É um tipo derivado de *TextWriter*. Na Figura 5 a seguir, está o formulário criado dentro do *Visual Studio* 2010. Há dois botões que irão executar as funções de leitura e escrita do arquivo. Na janela propriedade é possível observar o nome do botão e mais acima a estrutura do projeto no *Solution Explorer*. Tanto os botões quanto o *textbox* foram inseridos

clicando e arrastando os objetos, sem a necessidade da criação dos códigos necessários para sua exibição. A maior parte do trabalho ficou por conta do IDE.

Figura 5: Formulário em desenvolvimento.



Fonte: Elaborado pelos autores (2015).

Na Figura 6 é apresentado o código contido no botão "*Escrever Arquivo*".

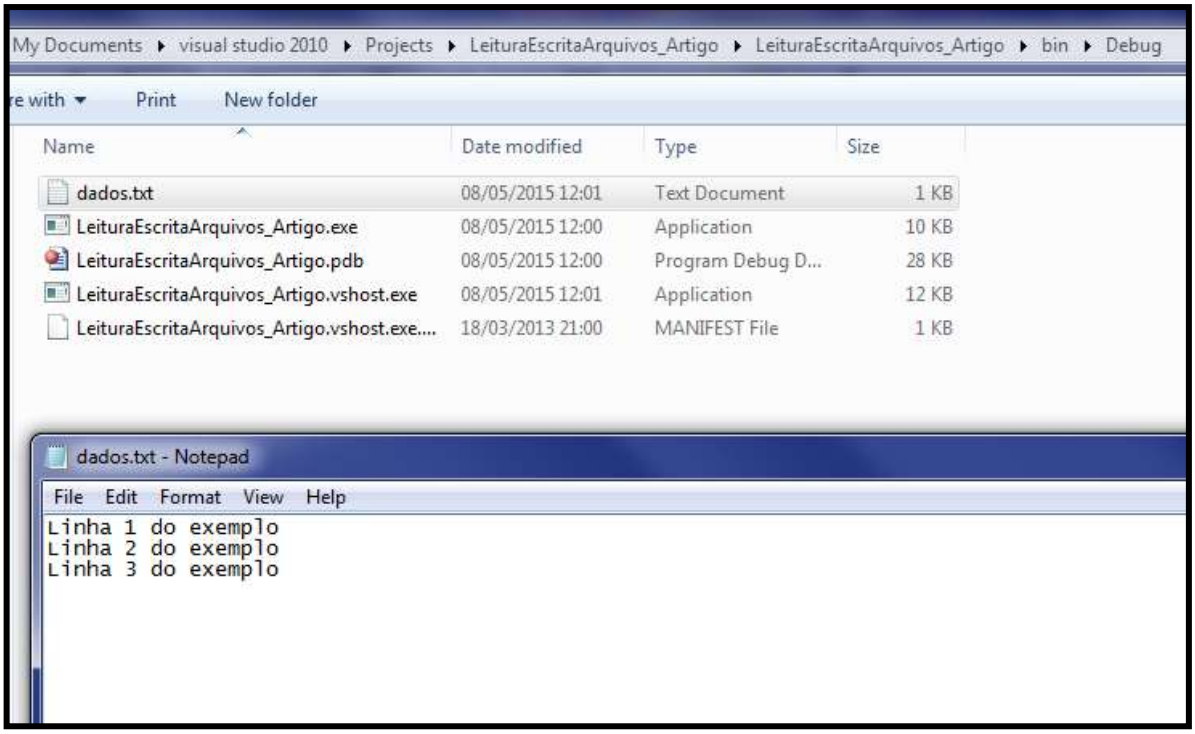
Figura 6: Código de escrita do arquivo.

```
private void btEscreve_Click(object sender, EventArgs e)
{
    using (StreamWriter escrever = new StreamWriter("dados.txt"))
    {
        escrever.WriteLine("Linha 1 do exemplo");
        escrever.WriteLine("Linha 2 do exemplo");
        escrever.WriteLine("Linha 3 do exemplo");
    }
}
```

Fonte: Elaborado pelos autores (2015).

Quando clicado no botão, a ferramenta irá gerar o arquivo "*dados.txt*" na pasta raiz do projeto. Dentro do arquivo serão criadas três linhas de texto, como pode ser observado nas três linhas similares na Figura 6. Cada linha dentro da instância é responsável pela escrita de uma linha no arquivo. Quando criado a variável "*escrever*" do tipo *StreamWriter*, já será possível usar seus atributos como o *WriteLine*, que é o atributo encarregado de escrever o texto desejado e pular uma linha. Na Figura 7 a seguir exibe os dados e arquivo gerado pelo código da Figura 6 anterior.

Figura 7: Código de escrita do arquivo.



Fonte: Elaborado pelos autores (2015).

É possível observar os arquivos gerados pelo *Visual Studio* dentro da pasta raiz do projeto. Lá estão os arquivos de propriedades do projeto, e o mais importante, o arquivo executável do projeto. Como não foi definido um caminho para o arquivo "*dados.txt*", ele é criado na pasta em que o arquivo executável do projeto se encontra. E acessando o arquivo pelo bloco de notas, observa-se que as três linhas, que foram criadas no projeto pela variável "*escrever*", estão contidas no arquivo de texto.

3.2 Leitura de Arquivos – *StreamReader*

Segundo (MICROSOFT, 2015), o *StreamReader* é a função que irá efetuar a leitura do arquivo que o sistema criou, no caso o "*dados.txt*". É um tipo derivado de *TextReader*. Na Figura 8 a seguir o código do botão "Ler Arquivo".

Figura 8: Código de leitura do arquivo.

```
private void btLer_Click(object sender, EventArgs e)
{
    using (StreamReader leitura = new StreamReader("dados.txt"))
    {
        txtLeitura.Text = leitura.ReadToEnd();
    }
}
```

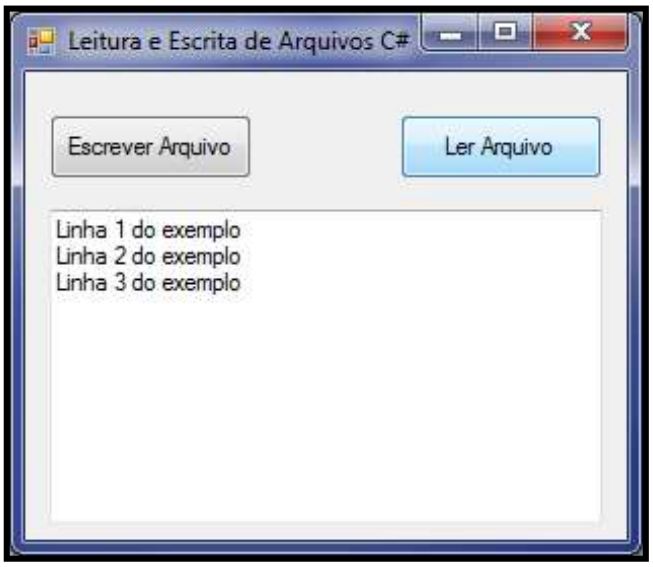
Fonte: Elaborado pelos autores (2015).

Quando é iniciada a variável "*leitura*", já é fornecido o nome do arquivo para leitura, assim o sistema já irá identificar e abrir o "dados.txt", que está localizado no mesmo diretório da aplicação. Logo após, o atributo "*ReadToEnd*" efetua a leitura de todas as linhas do arquivo até que chegue ao fim. Sendo assim, quando este atributo é chamado, ele percorre todas as linhas existentes no arquivo apontado e carrega exatamente os dados dentro da caixa de texto "*txtLeitura*". Há várias maneiras em que o usuário pode percorrer o arquivo manualmente linha por linha, mas pode-se observar que a linguagem C# oferece facilidades no trabalho.

A Figura 9 ilustra a aplicação compilada e em modo de execução no sistema operacional *Microsoft Windows*. Pode-se observar que há três linhas na caixa de texto (*textbox*). Elas foram carregadas pelo comando representado Figura 8. Ou

seja, quando clicado no botão "*Ler Arquivo*" o código carrega os dados do arquivo para dentro da *textbox*.

Figura 9: Código de leitura do arquivo.



Fonte: Elaborado pelos autores (2015).

CONSIDERAÇÕES FINAIS

Como observado tanto no estudo, nos exemplos e nas características acima abordadas, o C# é uma linguagem que foi bem estruturada em sua criação. Todos os exemplos demonstrados foram desenvolvidos de maneira rápida devido ao modo gráfico que simplifica o desenvolvimento do sistema apresentado. Uma limitação encontrada no C# é que a linguagem necessita do .NET *Framework* instalado, além disso, há dificuldades em rodar aplicações em *Linux*, onde serão coletadas mais informações para prosseguir com as pesquisas iniciadas.

O C#, a plataforma .NET e o *Visual Studio* da *Microsoft* podem ajudar aos desenvolvedores de programas a criarem mais rapidamente seus projetos. Isso pode gerar um retorno monetário de maneira breve e uma satisfação maior dos clientes. Acredita-se que essas características são importantes, pois, nos dias de hoje, o mercado de automação comercial e industrial cresce rapidamente, exigindo qualidade e agilidade.

NOTAS

[1] **XML:** *Extensible Markup Language.* É uma linguagem de marcação para padronização de dados e integração de conteúdos com outras linguagens.

[2] **SOAP:** *Simple Object Access Protocol.* É um protocolo que faz a transferência de mensagens no formato XML para uso em plataformas distribuídas.

[3] **COM:** *Component Object Model.* É uma plataforma da Microsoft que permite a comunicação entre aplicações de diferentes linguagens de programação.

[4] **ASP.NET:** *Active Server Pages. É uma linguagem de programação que foi criada para processar seu código pelo servidor e gerando conteúdo dinâmico na web.*

[5] **F#:** Linguagem de programação. É uma linguagem de multiparadigma, funcional e de programação imperativa. Totalmente direcionada a plataforma .NET.

[6] **VB.NET:** *Visual Basic .NET. Linguagem de programação criada pela Microsoft, e integrada ao pacote do software Visual Studio.*

REFERÊNCIAS

DEITEL, H. M.; *et al*. **C# - Como Programar**. São Paulo: Pearson Education, 2003.

IC#CODE. **IC#Code**. Disponível em: <http://www.icsharpcode.net/>. Acesso em: 05 jun. 2015.

LIMA, E. **C# e .Net para Desenvolvedores.** Rio de Janeiro: Campus, 2002.

MANZANO, J. A. N. G. **Estudo Dirigido de *Microsoft Visual C# Express* 2010.** São Paulo: Editora Érica, 2012.

MICROSOFT. **Microsoft Home Page Oficial**. Disponível em: <https://www.microsoft.com/pt-br/default.aspx>. Acesso em: 24 jun. 2015.

STELLMAN, A.; GREENE, J. **Use a Cabeça! C#**. Rio de Janeiro: Alta Books, 2008.

STIEFEL, M.; OBERG, R. J. ***Development Using C# and .NET**. United States of America: Prentice Hall PTR,* 2002.