FIAP ON

DESENVOLVIMENTO .NET

INTRODUÇÃO À PLATAFORMA .NET

FLAVIO MORENI

1

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 INTRODUÇÃO À PLATAFORMA .NET

1.1 História

Na década de 1990, a Microsoft tinha como produto principal as linguagens Visual Basic e Visual C++, que possuíam suporte de execução apenas na plataforma Windows. No final dessa década, iniciou-se a aceitação de linguagens independentes de plataforma de execução, tendo o Java como uma das mais conhecidas e usadas.

Com o nome de *Next Generation Windows Services* (NGWS), a Microsoft iniciou o desenvolvimento do .NET Framework, que foi lançado na sua versão beta no final do ano de 2001. O lançamento final da versão 1 do framework aconteceu meses depois, em fevereiro de 2002.

1.2 Criação do CSharp (C#)

Com o avanço das linguagens de programação (entre elas, o Java e o Delphi) e de dispositivos eletrônicos, as linguagens de programação foram obrigadas a criar recursos de execução para diversos dispositivos e plataformas. Como primeira estratégia, a Microsoft adotou a linguagem Java com o nome de J++, em um acordo de licenciamento com a Sun MicroSystems para o uso da linguagem na plataforma Windows. Esse acordo não foi suficiente, pois a exigência da época era executar essa linguagem em múltiplas plataformas e dispositivos.

Assim, a nova estratégia da empresa foi a criação de uma nova linguagem independente de licenciamentos e acordos, com grande foco em independência de plataforma e dispositivo. Essa iniciativa foi criada a partir do projeto COOL (C-like Object Oriented Language), que teve como base outras linguagens, como:

- Java.
- C.
- C++.
- Smalltalk.
- Delphi.

- Visual Basic (VB).

O projeto COOL foi renomeado para C# 1.0 (C Sharp 1.0) e lançado em conjunto com o .NET Framework em 2002. Desde essa época, a linguagem passou por várias atualizações, a versão mais atual é a 7.0, que teve como uma das grandes melhorias a implementação de chamadas assíncronas, além da evolução na velocidade de execução de comandos.

O Framework .NET é um ambiente para a execução de programas de computador que fornece uma variedade de serviços aos aplicativos em execução.

Seus principais componentes são:

- CLR (Common Language Runtime), responsável por gerenciar a execução de aplicativos.
- Biblioteca de classes, responsável em prover uma coleção de componentes e códigos que os desenvolvedores possam usar em seus softwares.

Em resumo, o Framework .NET é um ambiente de execução de código e bibliotecas usado por desenvolvedores para criar e executar programas. Assim como o Java e outras linguagens, utiliza do conceito de máquina virtual, que cria uma camada entre o sistema operacional e a aplicação.

O componente responsável por esse isolamento entre aplicação e sistema operacional é o CLR (*Common Language Runtime*). Porém, o CLR possui mais responsabilidades do que somente o isolamento entre aplicação e sistema operacional. O CLR também é responsável por:

- Gerenciamento de memória.
- Tipos comuns de variáveis.
- Bibliotecas para tipos exclusivos de projetos (por exemplo: projetos de internet, acesso a banco de dados, projetos de aplicativos móveis e outros).
- Compatibilidade de versão.
- Multiplataforma (por exemplo: Windows 7, Windows 8, Windows 8.1, Windows 10, Windows Phone e Xbox 360).

- Execuções paralelas com diferentes versões do framework.

Abaixo, segue a imagem com a evolução do .NET Framework, do seu lançamento até a versão 4.5 de 2012.

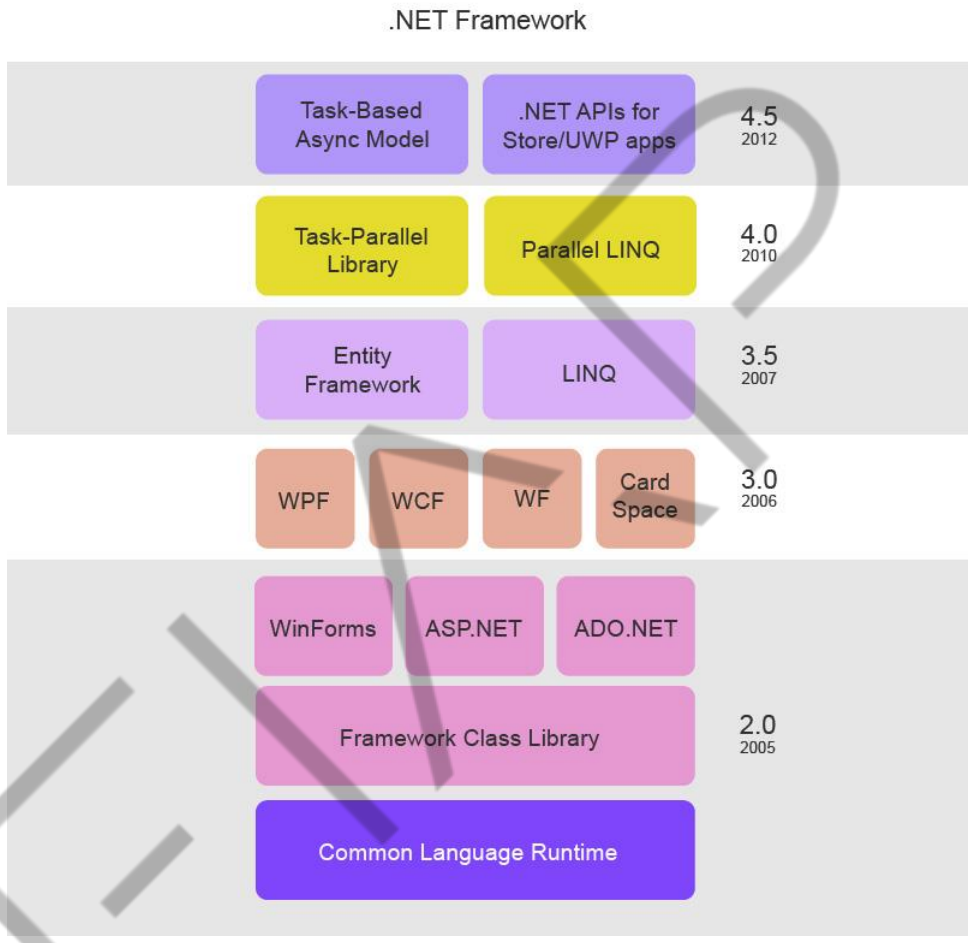

Figura 1.1 Histórico de evolução do .NET Framework
Fonte: Google Imagens (2018)

1.3 Visual Studio

O ambiente de desenvolvimento ou IDE (Integrated Development Environment) do .NET Framework é chamado de Visual Studio. É a ferramenta de suporte ao desenvolvimento das linguagens C# (C Sharp), Visual Basic .NET (VB.NET), C, C++ e Xamarin.

O Visual Studio é uma ferramenta completa para desenvolvimento que possui um grande suporte para o desenvolvimento de websites, serviços web e aplicativos

móveis. Oferece aos desenvolvedores suporte a bibliotecas, como ASP.NET MVC, para desenvolvimento de aplicações web; Xamarin.Forms, para aplicativos de celular na plataforma Android; e IOS e Windows.Forms, para desenvolvimento de aplicações para desktop.

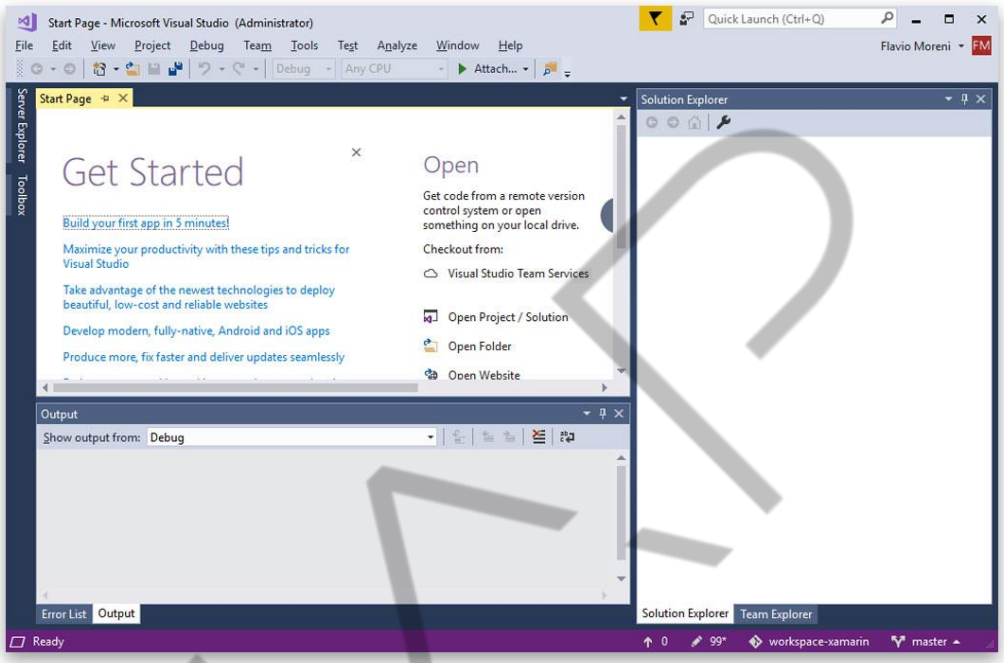

Figura 1.2 – Tela inicial do Visual Studio 2017
Fonte: Elaborado pelo autor (2018)

1.3.1 Obtendo o Visual Studio

A versão mais atual do Visual Studio *Community* pode ser baixada no site: <https://www.visualstudio.com/pt-br/downloads/>. O site possui também alguns materiais de referência da ferramenta, histórico e dicas e pré-requisitos do sistema para instalação da ferramenta.

DICA: Os alunos da FIAP têm direito de usar a versão *Enterprise* sem nenhum custo. Para isso, basta acessar o portal do aluno, selecionar o menu benefícios e acessar a opção **DreamSpark**.

Para usuários de computadores Apple, uma versão do Visual Studio pode ser baixada em: <https://www.visualstudio.com/pt-br/vs/visual-studio-mac/>.

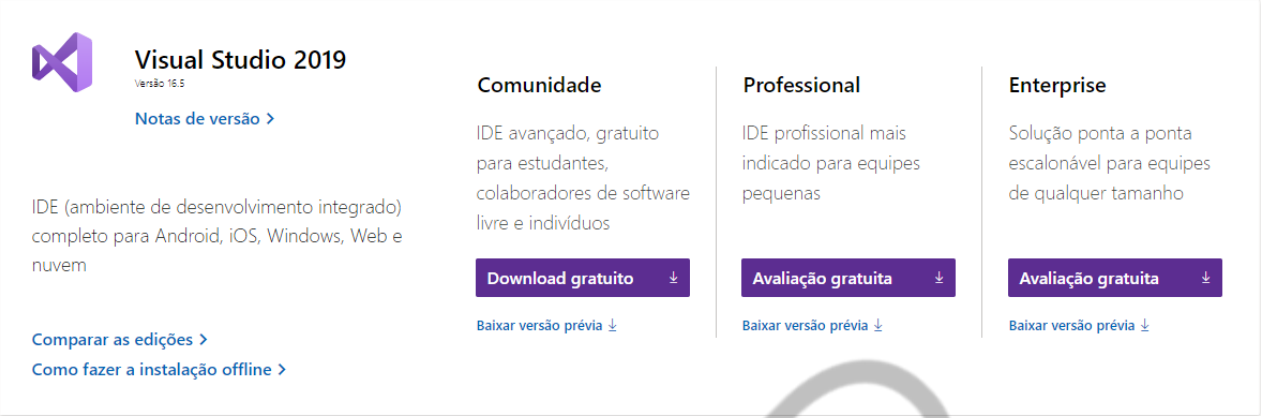

Figura 1.3 – Download de Visual Studio 2019 e documentação
Fonte: visualstudio.com (2020)

1.3.2 Produtos suportados

OVisual Studio oferece suporte a três linguagens de programação, são elas:

- Microsoft Visual C++.
- Microsoft Visual C#.
- Microsoft Visual Basic (VB.NET).

Possui também o Microsoft Visual Web Developer, plataforma para desenvolvimento de aplicações ou serviços web, para os quais se deve usar as bibliotecas do pacote ASP.NET, podendo-se escolher entre as linguagens C# ou VB.NET.

Para controlar os códigos-fonte, gerenciar o desenvolvimento e criar relatórios de produtividade, o Visual Studio oferece suporte ao Team Foundation Server (TFS).

E, por fim, o VS 2017 possibilita a instalação da plataforma de desenvolvimento de aplicativos móveis no formato híbrido (Android e IOS), chamado de Xamarin.Forms.

As primeiras versões do Visual Studio ofereciam suporte para as linguagens de programação Visual FoxPro e Microsoft Visual J++/J#. Tinham como ferramenta de gerenciamento de código-fonte o Visual SourceSafe e também a ferramenta Visual InterDev, para auxiliar na criação de aplicações Microsoft Active Server Pages (ASP).

1.4 Projetos

1.4.1 Estrutura do projeto

O espaço de trabalho para o desenvolvedor no Visual Studio é conhecido como *solution*, ou “solução”, em português. No sistema de arquivos do computador, toda solução é apresentada por arquivo com a extensão sln. Podemos fazer uma comparação entre o arquivo .sln (Visual Studio) e a pasta *workspace* ou o arquivo. project da IDE eclipse (Java), que possuem diferentes formatos, mas o mesmo objetivo, ou seja, armazenar seu espaço de trabalho e projetos.

A solução é responsável por agrupar vários projetos .NET, permitindo a navegação entre eles e a compilação de todos ao mesmo tempo.

O projeto .NET é o responsável pelo agrupamento do código-fonte, ícones, imagens, xml, dll e qualquer outra fonte que será compilada. No sistema de arquivo, um projeto é apresentado pela extensão .csproj (C#) ou .vbproj (VB).

Veja abaixo a estrutura da solução e projetos no Visual Studio e no sistema de arquivos do Windows:

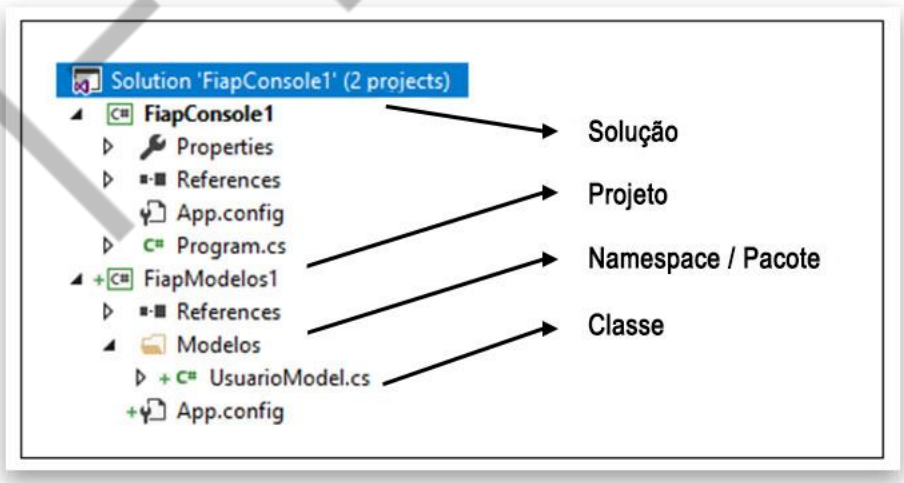

Figura 1.4 – Estrutura da solução no Visual Studio
Fonte: Elaborado pelo autor (2018)

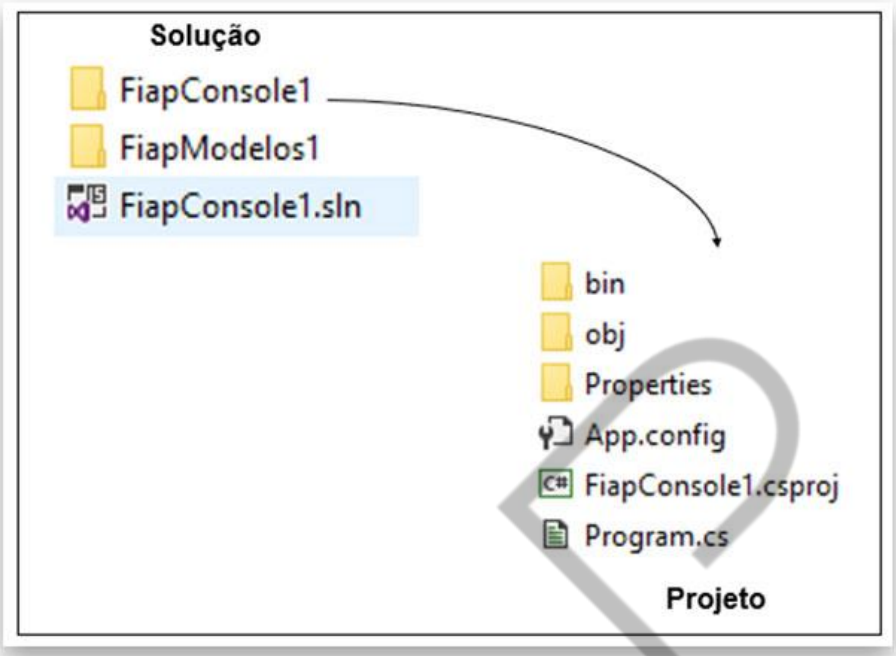

Figura 1.5 – Estrutura da solução no sistema de arquivos Windows
Fonte: Elaborado pelo autor (2018)

Diferentemente da plataforma Java, que deixa no seu espaço de trabalho diversos projetos de domínios de negócio totalmente diferentes e sem ligação entre eles, a plataforma .NET sugere que na solução (espaço de trabalho) sejam criados apenas projetos do mesmo domínio de negócio ou que tenham ligação entre si. Assim, cada solução criada no Visual Studio deve representar um contexto de negócio.

Atente-se para a criação de vários projetos em uma única solução, essa estratégia faz sentido apenas quando um projeto será usado em um grupo de projetos. Em muitos casos, a estratégia de criação de pacotes (*namespaces*) resolve o problema de criação de muitos projetos em uma solução que tenha como objeto o agrupamento de funcionalidade, camadas e componentes.

1.4.2 Criando um projeto

Agora que sabemos um pouco sobre a história do Framework .NET e conhecemos mais sobre a IDE Visual Studio, vamos criar o primeiro programa

utilizando o Visual Studio e aprofundar nosso conhecimento no uso da IDE e da plataforma .NET.

Para o primeiro programa, será necessária a criação de uma solução de projeto. Com o Visual Studio aberto, selecione o menu *File* > *New* > *Project* (a tecla de atalho Crtl + Shift + N), para abertura da tela de criação de novos projetos.

A Figura Tela de criação de projeto apresenta a tela inicial de criação de um projeto e os *templates* disponíveis.

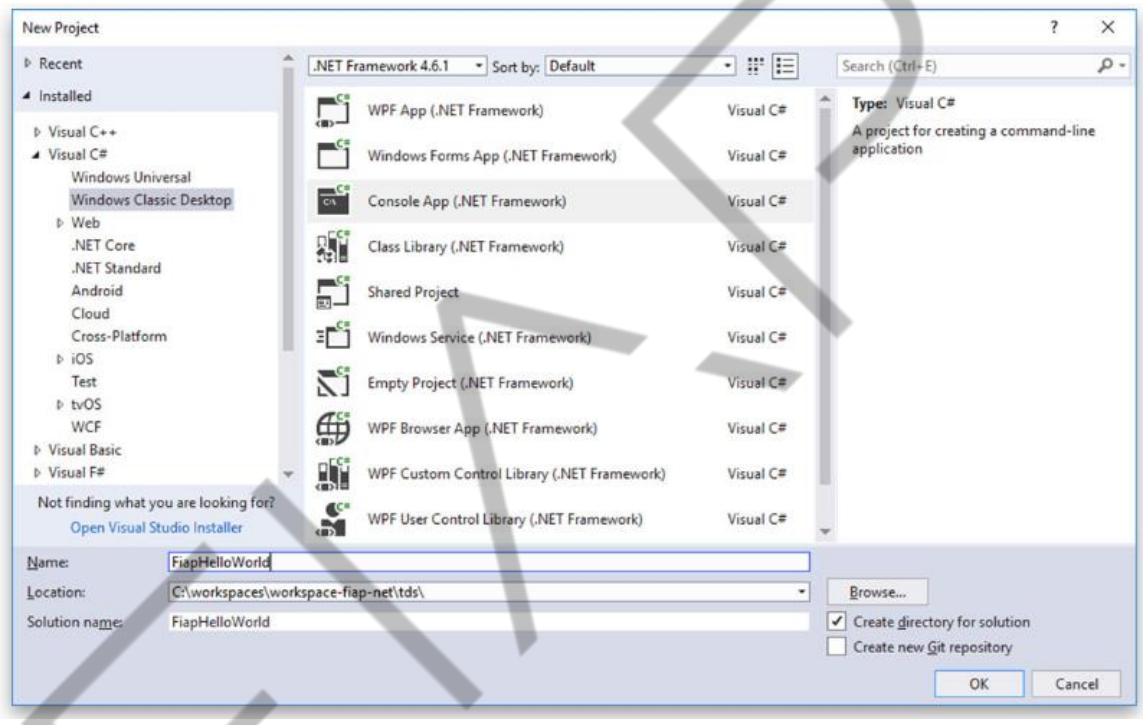

Figura 1.6 – Tela de criação de projeto
Fonte: Elaborado pelo autor (2018)

Na tela de criação, devemos escolher a linguagem Visual C# na parte direita da janela. No centro, vamos selecionar o tipo de projeto Console App (.NET Framework). Na parte inferior, temos a caixa de texto para definir o nome do projeto, o local no sistema de arquivos e o nome da solução. Para nosso exemplo, vamos usar **FiapHelloWorld** como nome do projeto e da solução.

A Figura Projeto FiapHelloWorld apresenta o projeto aberto no Visual Studio e sua classe **Program.cs** com conteúdo aberto no editor.

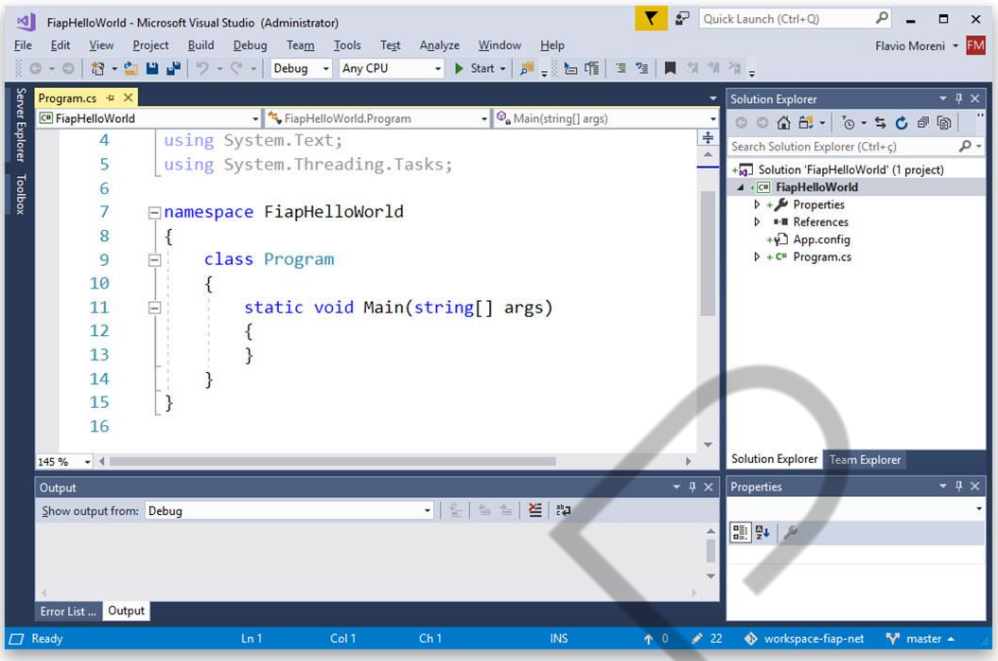

Figura 1.7 – Projeto FiapHelloWorld
Fonte: Elaborado pelo autor (2018)

Com o projeto aberto, e por meio do **Solution Explorer** posicionado no lado esquerdo da janela do Visual Studio, podemos navegar para o sistema de arquivos e verificar os arquivos do projeto. Com um clique do botão direto em cima do projeto ou da solução, selecione a opção **Open Folder in File Explorer**.

1.4.2.1 Escrevendo o código

Anteriormente, foram executados os passos para a criação de um projeto C# do tipo ***Console Application***, que por padrão gera a classe **Program.cs**, a qual será responsável pela execução dos nossos comandos.

Neste tópico, vamos inserir algumas linhas de código e executar o primeiro programa. Com um duplo clique no arquivo *Program.cs*, disponível na janela ***Solutions Explorer***, podemos editar o conteúdo do programa e inserir as linhas de código necessárias. Assim, vamos inserir uma linha para a impressão de uma mensagem na tela e outra linha que mantém a janela para a visualização do usuário até que uma tecla seja pressionada.

O código-fonte deverá ficar como o do exemplo a seguir:

```
using System;

namespace FiapHelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Fiap - Ola C#");

            // trecho para manter a janela aberta
            Console.Read();
        }
    }
}
```

Código-fonte 1.1 – Primeiro programa no Visual Studio
Fonte: Elaborado pelo autor (2017)

DICA: Atente-se para o método ***Main*** e as linhas inseridas dentro dele.

1.4.2.2 Compilando e executando

A forma mais simples e prática para executar uma aplicação criada no Visual Studio é pressionando a tecla **F5**. Com o projeto **FiapHelloWorld** aberto, pressione a tecla **F5** e observe o resultado, conforme a Figura Janela de execução do aplicativo console a seguir:

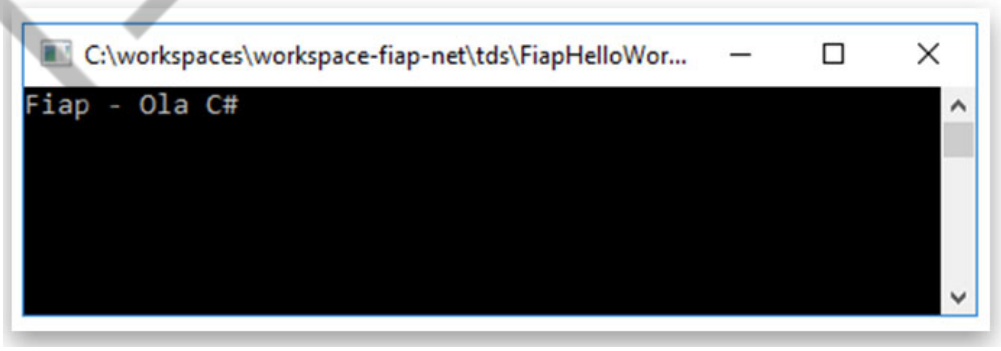

Figura 1.8 – Janela de execução do aplicativo console
Fonte: Elaborado pelo autor (2018)

Para encerrar a execução do programa, na tela de exibição da mensagem “Fiap - Ola C#”, pressione a tecla Enter, ou na janela do Visual Studio aperte “Shift + F5”.

A tecla F5 é a forma mais simples de compilação e execução de projetos, mas é possível realizar outras ações no Visual Studio para facilitar o dia a dia do desenvolvedor, como: compilar todos os projetos de uma solução, compilar apenas um projeto sem executá-lo, limpar compilações anteriores e até efetuar uma análise do código criado. Essas ações podem ser acionadas pelo menu ***Build***, na barra superior da ferramenta.

Outro atalho para compilação e execução é a barra de tarefas ***Standard***, que apresenta botões para execução, encerramento, continuação em caso de *debug* e outras funções não relacionadas à execução e compilação. A seguir, é possível visualizar os botões de atalho para compilação e execução:

Figura 1.9 – Barra de ferramentas compilação e execução
Fonte: Elaborado pelo autor (2018)

1.4.2.3 Organizando o projeto

Para organizar nossos projetos em C#, precisamos falar de *namespaces*. Essa palavra é reservada no C#, responsável por declarar um escopo ou bloco que contém um conjunto de classes relacionadas. Também pode ser usada para controlar o acesso entre conjunto de classes de *namespaces* diferentes. Uma associação comum usada para explicar o conceito de *namespace* é o conceito de *packages* em Java, com a diferença de que um *package* Java deve seguir o mesmo nome da estrutura de diretório; em contrapartida, no C# o caminho ou o nome do *namespace* não precisa respeitar o caminho do diretório dos componentes.

Por padrão, o Visual Studio define como primeiro *namespace* o nome do projeto.

Veja o exemplo da Figura *Namespace-padrão* criado pelo Visual Studio:

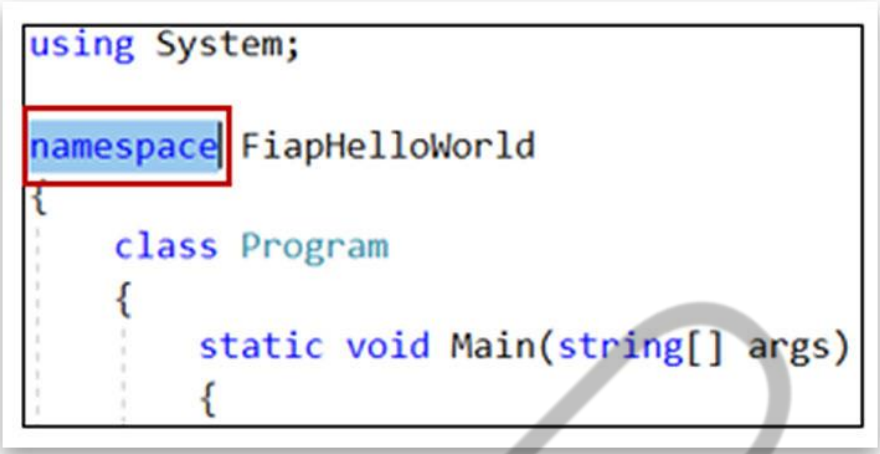

Figura 1.10 – *Namespace-padrão* criado pelo Visual Studio
Fonte: Elaborado pelo autor (2018)

O padrão definido pela Microsoft para a criação de *namespaces* deve seguir o do exemplo abaixo:

NomeDaEmpresa.NomeDoProjeto.ModuloDoSistema

E outra boa prática que vamos adotar é a criação das pastas na estrutura do nome usado para o *namespace.*

Exemplo: **...\FiapHelloWorld\FiapHelloWorld\Models**

Segue na Figura Exemplo de um *namespace* um exemplo da estrutura de pasta e *namespace*.

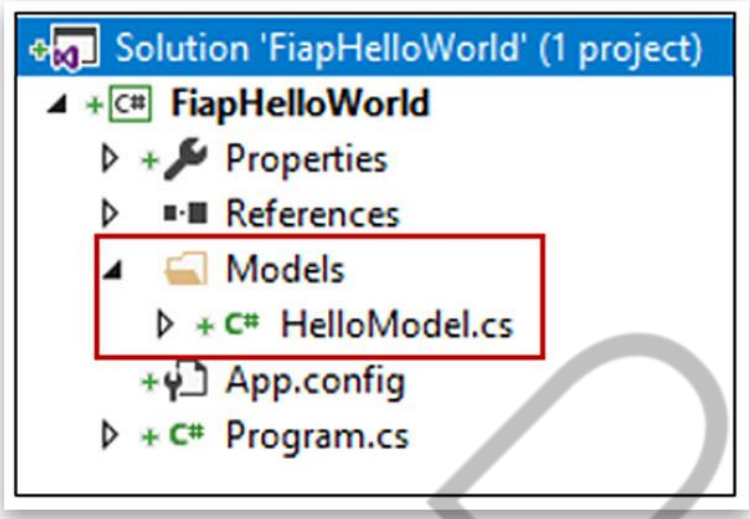

Figura 1.11 – Exemplo de um *namespace*
Fonte: Elaborado pelo autor (2018)

Para a criação de um *namespace*, basta clicar com o botão direito no projeto C#, escolher a opção *Add* > *New Folde*r e digitar o nome da pasta. Para o nosso exemplo, será criada uma pasta com o nome **Models**. Para entender o uso do *namespace*, clique com o botão direito na pasta Models e selecione a opção *Add*> *Class*. Selecione a opção Class e defina o nome de **HelloModel**, como pode ser visto na Figura Adicionando nova classe ao *namespace* a seguir:

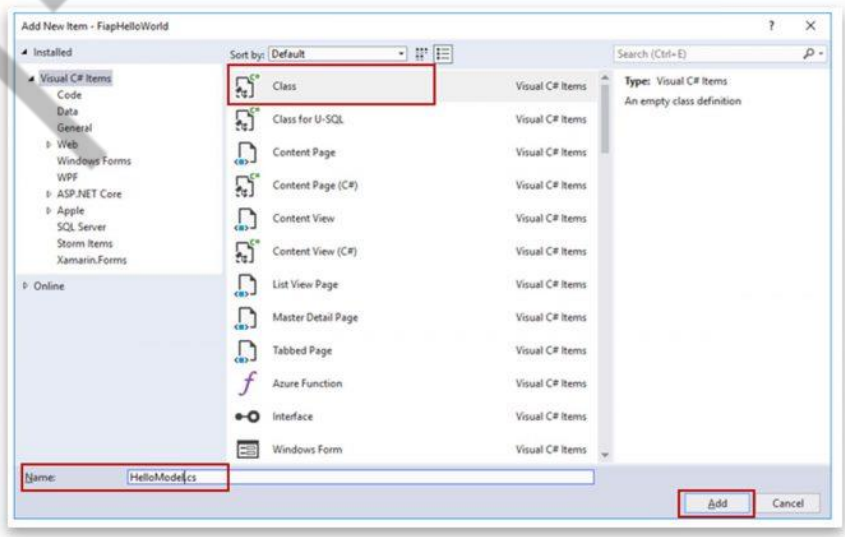

Figura 1.12 – Adicionando nova classe ao *namespace*
Fonte: Elaborado pelo autor (2018)

Analisando o código-fonte da classe criada, pode-se notar que foi definido o *namespace* FiapHelloWorld.Models como padrão. Pois bem, é possível definir outro nome para um *namespace* sem mudar o nome da pasta, porém, por questão de bom senso, vamos mantê-los sempre iguais.

Dentro de um *namespace,* é possível criar, além de classes, outros tipos de componentes, como: *interface, struct, enum, delegate.*

Vamos alterar nosso exemplo e fazer o uso da classe de modelo e validar a organização do projeto. Para isso, vamos definir uma propriedade de mensagem na classe de modelo, conforme o Código-fonte 7.2 abaixo:

```
using System;

namespace FiapHelloWorld.Models
{
    public class HelloModel
    {
        public String Mensagem = "Ola Model C#";
    }
}
```

Código-fonte 1.2 – Classe de exemplo do *namespace* Models
Fonte: Elaborado pelo autor (2018)

Agora vamos alterar a classe **Program.cs** para acessar a classe modelo e imprimir o texto declarado na propriedade Mensagem.

```
using System;

namespace FiapHelloWorld
{
    class Program
    {
       static void Main(string[] args)
       {
       Models.HelloModel model = new Models.HelloModel();
       Console.WriteLine(model.Mensagem);
       Console.Read();
       }
    }
}
```

Código-fonte 1.3 – Usando uma classe de outro *namespace*
Fonte: Elaborado pelo autor (2018)

Basta executar o programa e verificar a nova mensagem na tela.

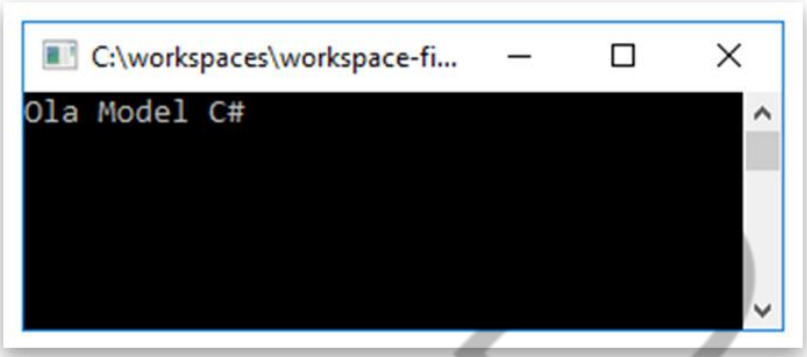

Figura 1.13 – Tela de verificação de exemplo de *namespace*
Fonte: Elaborado pelo autor (2018)

1.4.2.4 Debug

Até este ponto, conseguimos executar uma aplicação usando o Visual Studio e a linguagem C#. Agora temos a necessidade de validar alguns pontos do código e navegar pelas classes do projeto durante a execução a fim de entender o que acontece com nossas classes, atributos e comandos. Para isso, vamos usar as ferramentas do Visual Studio para depuração.

Vamos iniciar pela classe do modelo (**Models\HelloModel.cs**). Abra a classe no editor e posicione o cursor na linha de criação do atributo Mensagem. Com a tecla **F9** ou um clique na margem esquerda da janela do editor, podemos adicionar um ponto de interrupção (*breakpoint*). A Figura Ponto de interrupção (*breakpoint*) apresenta a linha selecionada e o ponto de interrupção criado.

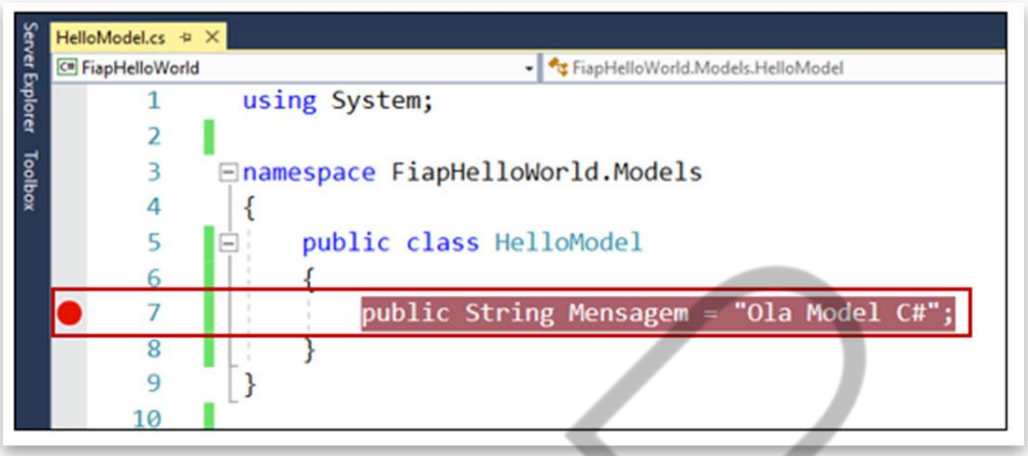

Figura 1.14 – Ponto de interrupção (*breakpoint*)
Fonte: Elaborado pelo autor (2018)

Para testar o *breakpoint*, execute o projeto (F5) e espere até o Visual Studio interromper a execução ao chegar na linha selecionada. Com a execução interrompida na linha selecionada, algumas ações para ajudar na compreensão do programa podem ser tomadas. A seguir, vamos ver detalhes das ações mais comuns de debug.

1.4.3 Immediate Window

É uma ferramenta para inserir comandos, mudar valores de variáveis ou testar regras em tempo de execução. A janela *Immediate Windows* é exibida no rodapé do Visual Studio no momento da execução do aplicativo, ou pode ser aberta no menu ***Debug > Windows > Immediate*** ou **Crtl + Alt + I**.

Dentro da janela, você pode inserir linhas de código C# para alterar valores ou acessar o conteúdo e verificar valores. A tecla enter executa a alteração.

A Figura Janela Immediate Window, a seguir, mostra a ferramenta immediate, uma linha de comando para alteração do conteúdo de uma propriedade e o resultado da alteração.

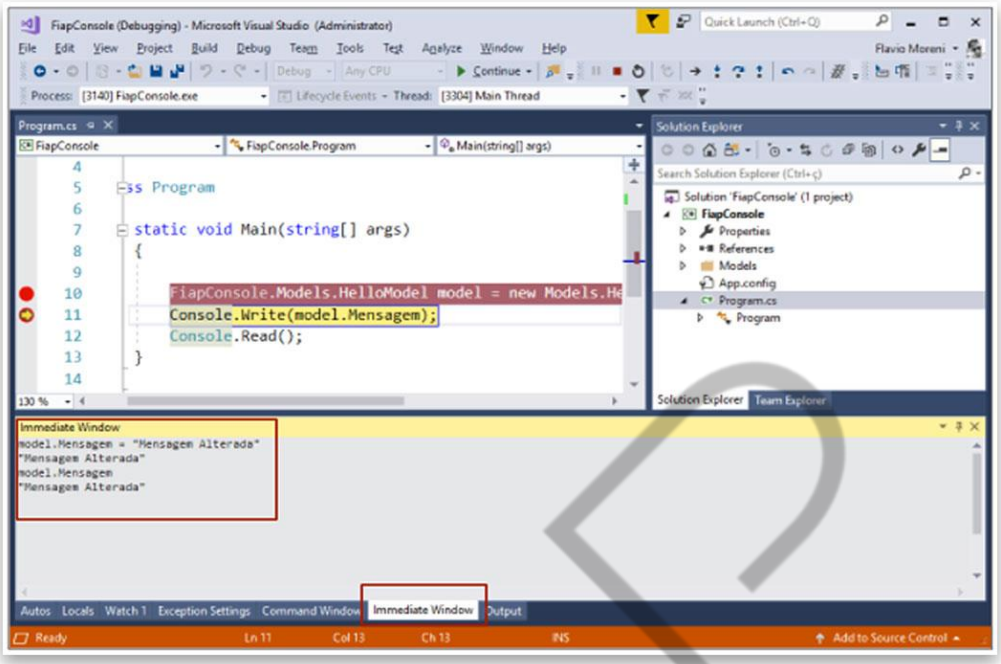

Figura 1.15 – Janela Immediate Window
Fonte: Elaborado pelo autor (2018)

1.4.4 Quick Watch

É a forma mais rápida de acessar conteúdos de variáveis, objetos ou expressões durante a execução em modo debug. A janela *Quick Watch* pode ser acessada clicando com o botão direito sobre a variável e selecionando a opção **QuickWatch…** ou pela tecla **Shift + F9**.

A Figura Janela Quick Watch, a seguir, apresenta a janela de inspeção, e objeto modelo do nosso exemplo, com seus valores em tempo de execução.

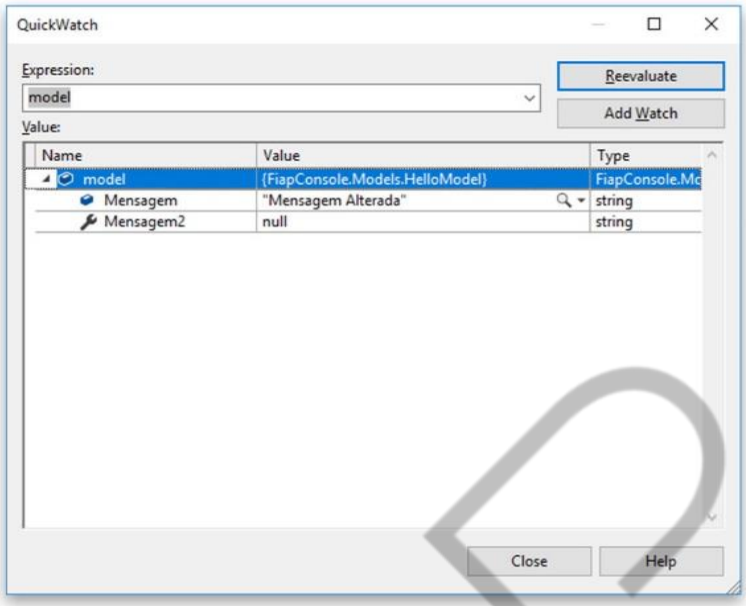

Figura 1.16 – Janela Quick Watch
Fonte: Elaborado pelo autor (2018)

1.4.5 Navegando pelo código

A lista abaixo aponta as formas possíveis de navegação por *breakpoints*, e suas teclas de atalho:

- *Step-Into* (F11), para executar a próxima linha de código, mesmo se a linha de código estiver em outro bloco de código, outra classe ou até mesmo em outra rotina externa.
- *Step-over* (F10), usado para avançar a execução dentro da mesma estrutura de código.
- *Step Backward* (Alt + ´), para retroceder a execução para a linha anterior.
- *Continue* (F5), executa a aplicação até o próximo *breakpoint.*
- *Step Out* (Shift + F11), usado para retornar ao ponto de debug que originou a chamada.

As ações de *Debug* podem ser usadas por meio das teclas de atalho, ou na opção *Debug* no menu superior, ou nos botões da barra de ferramenta. A Figura Barra

de ferramentas para *debug*, a seguir, apresenta as opções de *debug* na barra de ferramenta.

Figura 1.17 – Barra de ferramentas para *debug*
Fonte: Elaborado pelo autor (2018)

Para validar o uso das teclas, vamos adicionar um *breakpoint* na primeira linha de execução da classe **Program.cs** e executar o projeto (**F5**).

Com o programa parado no primeiro *breakpoint*, use a tecla **F11** para avançar e navegar para as linhas de código da classe de modelo **HelloModel**. A Figura Depurando passo a passo, a seguir, apresenta a linha de código da classe **HelloModel**, cuja execução o Visual Studio interrompeu. Note que a linha não possuía um *breakpoint* configurado.

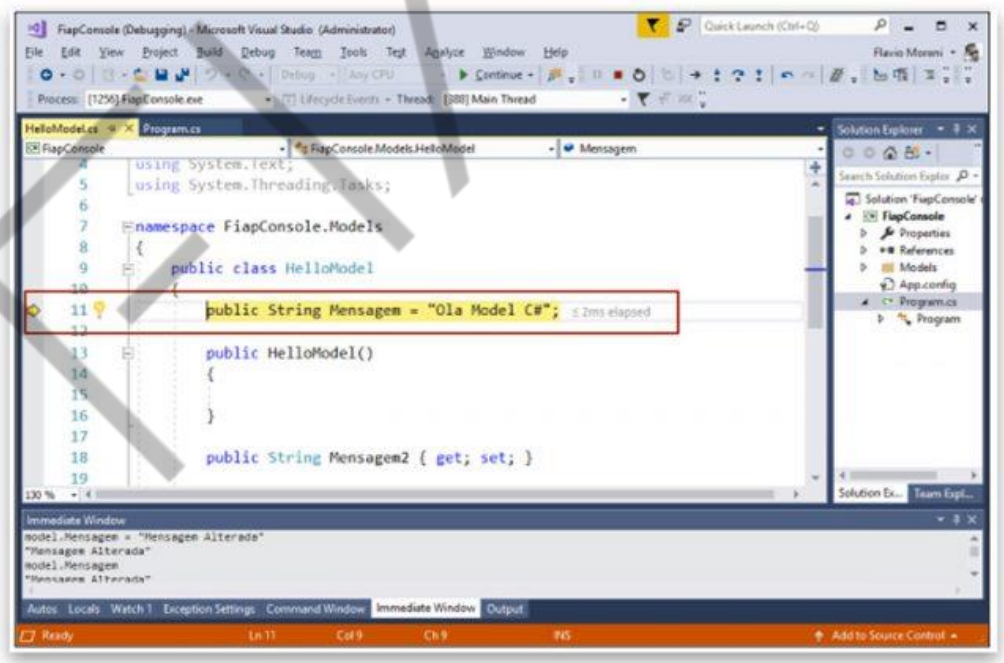

Figura 1.18 – Depurando passo a passo
Fonte: Elaborado pelo autor (2018)

De acordo com a necessidade, utilize as teclas F10, F11 ou F5 para avançar a execução, ou a combinação (Alt + ´) para retroceder as linhas.

DICA: O nível de complexidade do código dos exemplos é baixo, porém as soluções profissionais do dia a dia possuem algoritmos complexos. Por isso, o uso de *breakpoint* e a depuração tornam-se essenciais para o desenvolvedor.

1.4.6 Atalhos

Assim como as demais *ides*, o Visual Studio fornece diversas teclas de atalho para facilitar o dia a dia do desenvolvedor e aumentar a produtividade na construção de código. O Quadro “Atalhos do Visual Studio” mostra algumas opções mais comuns para os desenvolvedores C#.

Atalho	Descrição
Crtl + L	Remove a linha de código em que o cursor está posicionado.
Crtl + K, Crtl + D	Formata (identa) todo o código da classe em edição.
Crtl + Shift + B	Complicação de todos os projetos da solução.
Crtl + .	Abre a opção de *SmartTags,* para correções rápidas de código ou criação rápida de código. É usado também como *autocomplete*.
ctor + tab + tab	Cria o código do construtor da classe.
prop + tab + tab	Cria um atributo para a classe.
propfull + tab + tab	Cria um atributo para a classe e adiciona a implementação do *get* e *set*.
propg + tab + tab	Cria um atributo somente leitura, em que o *set* é declarado como privado.

Quadro 1.1 – Atalhos do Visual Studio
Fonte: FIAP (2016)

1.5 Considerações Finais

Este capítulo teve como objetivo apresentar a plataforma .NET, o resumo da história dessa tecnologia. Foi abordado de forma introdutória o *Framework .NET*, as linguagens suportadas por ele, os componentes e a criação da linguagem C Sharp (C#).

Foi apresentada a ferramenta Microsoft Visual Studio e a forma de obtenção, além das linguagens suportadas e os tipos de projetos que podem ser criados. Finalizando com a criação de um projeto simples para a familiarização com a IDE, organização, compilação e depuração.

REFERÊNCIAS

ARAÚJO, E. C. **C# e Visual Studio Desenvolvimento de aplicações desktop**. São Paulo: Casa do Código, 2015.

ARAÚJO, E. C. **Orientação a Objetos em C# – Conceitos e implementações em .NET**. São Paulo: Casa do Código, 2017.

CARDOSO, G. S. **Criando aplicações para o seu Windows Phone**. São Paulo: Casa do Código, 2014.

LIMA, E. **C# e .NET – Guia do Desenvolvedor**. Rio de Janeiro: Campus, 2012.

MICROSOFT. **Atalhos de teclado-padrão no Visual Studio.** Disponível em: <https://msdn.microsoft.com/pt-br/library/da5kh0wa.aspx>. Acesso em: 31 jan. 2018.

MICROSOFT. **Introdução ao .NET Framework.** Disponível em: <https://msdn.microsoft.com/pt-br/library/hh425099(v=vs.110).aspx>. Acesso em: 21 jan. 2018.

MICROSOFT. ***Namespace* (Referência de C#)**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/namespace>. Acesso em: 21 jan. 2018.

RODRIGUES, C. **Usando o debugger no ASP.NET**. Disponível em: <https://www.microsoft.com/brasil/msdn/Tecnologias/aspnet/DebuggerASPNET.M>. Acesso em: 16 jan. 2018.

GLOSSÁRIO

VS	Visual Studio.