FRAMEWORKS JAVA, .NET &
WEBSERVICES

A ALTERNATIVA .NET

2B

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 A ALTERNATIVA .NET

Embora tenha um imenso mercado e há décadas seja bastante presente e consolidada em produtos, a linguagem de programação Java não é a nossa única opção. Se você escolheu essa trilha de conhecimento e está interessado em uma alternativa, precisa conhecer a plataforma .NET da Microsoft.

Concebida sob vários aspectos à imagem e semelhança do Java, a plataforma .NET possui uma série de vantagens interessantes, conforme veremos nos próximos capítulos.

Seguir por esse caminho significa conhecer as duas principais linguagens de programação da atualidade, ampliando, em muito, as suas chances de participar de um grande projeto no mercado de trabalho. Vamos lá?

1.1 História

Na década de 1990, a Microsoft tinha como produtos principais as linguagens Visual Basic e Visual C++, ambas possuíam suporte de execução apenas na plataforma Windows. No fim dessa mesma década, iniciou-se a aceitação de linguagens independentes de plataforma de execução, sendo o Java uma das mais conhecidas e utilizadas com essas características.

Com o nome de *Next Generation Windows Services* (NGWS), a Microsoft iniciou o desenvolvimento do .NET Framework, que foi lançado em sua versão beta no fim 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, Java e Delphi) e dos 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 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 chamado 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 então, a linguagem passou por várias atualizações. A versão mais atual é a 7.0, que teve como uma de suas grandes melhorias a implementação de chamadas assíncronas, além da evolução na velocidade de execução de comandos.

1.3 .NET Framework

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

Seus componentes principais são:

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

Em resumo, o .NET Framework é um ambiente de execução de código e de bibliotecas usado por desenvolvedores para criar e executar programas. Assim como

o Java e outras linguagens, utiliza o 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 *Common Language Runtime* (CLR), porém, o CLR possui mais responsabilidades do que somente o isolamento entre aplicação e sistema operacional. O CLR é responsável também 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.

Veja, a seguir, a Figura “Histórico de evolução do .NET Framework”, com a evolução do .NET Framework desde o seu lançamento até a versão 48, de 2019.

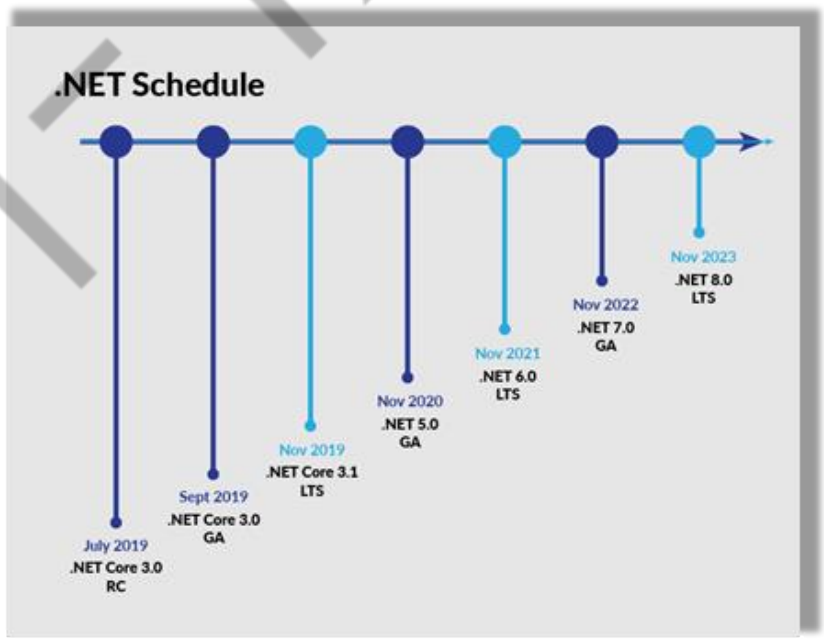

Figura 1 – Histórico de evolução do .NET Framework
Fonte: Microsoft (2022)

2 VISUAL STUDIO

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

O Visual Studio é uma ferramenta completa para desenvolvimento, 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 nas plataformas Android e iOS; Windows.Forms para o desenvolvimento de aplicações para desktop.

Versões do Visual Studio IDE:

- **Community** - A edição Community foi anunciada em 12 de novembro de 2016 como uma versão nova e gratuita com funcionalidade similar ao Visual Studio Professional. Visual Studio Community dá suporte a múltiplas linguagens e fornece suporte para extensões. É orientado para desenvolvedores individuais e pequenas equipes.
- **Professional** - Fornece aos desenvolvedores individuais e às pequenas equipes uma solução completa que inclui um ambiente de desenvolvimento integrado (IDE) de 64 bits. O que os ajuda a codificar, colaborar e enviar de forma mais rápida e segura entre idiomas e tipos de projetos e integrar segurança e DevOps em todo o ciclo de vida do desenvolvimento.
- **Enterprise** - Em adição às funcionalidades fornecidas pela edição Professional, a edição Enterprise fornece um novo grupo de ferramentas para desenvolvimento de software, desenvolvimento de banco de dados, colaboração, métricas, arquitetura, testes e relatórios.

Outras versões do Visual Studio:

- **Visual Studio Code:** É um editor de código-fonte leve, mas poderoso, que é executado em sua área de trabalho e está disponível para Windows, macOS e Linux. Ele vem com suporte integrado para JavaScript, TypeScript e Node.js, tem um rico ecossistema de extensões para outras línguas e tempos de execução (como C++, C#, Java, Python, PHP, Go, .NET).
- **Visual Studio For Mac:** Para usuário do sistema operacional MacOS (Apple), ele adota totalmente a experiência do macOS com controles nativos em todo o IDE, um novo modo escuro e ferramentas de acessibilidade nativas do macOS.

- **DICA:** Os alunos FIAP têm direito a usar a versão *Enterprise* sem nenhum custo. Para isso, basta acessar o portal do aluno, selecionar o menu benefícios e utilizar a opção que dá acesso ao portal da Microsoft.

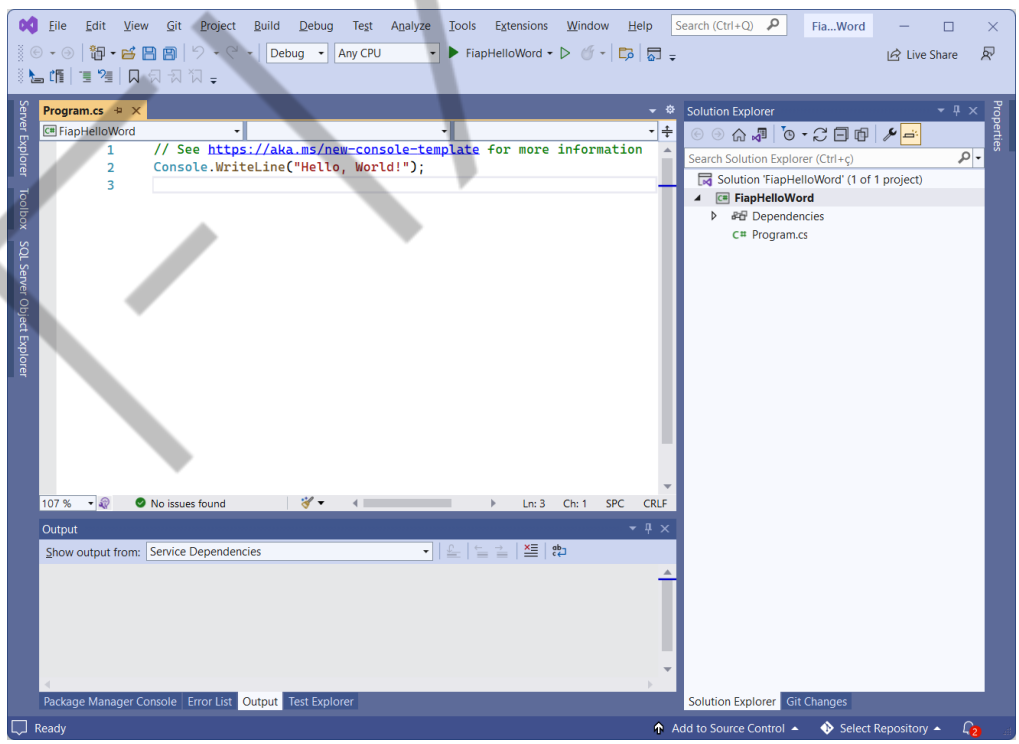

Figura 2 – Tela inicial do Visual Studio
Fonte: Elaborado pelo autor (2022)

2.1 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, dicas e pré-requisitos do sistema para instalação da ferramenta.

DICA: Até a edição desse conteúdo a última versão da IDE é o **Visual Studio 22** versão **17**. E em toda jornada será utilizado o **Runtime .NET 6.0**.

Para usuários de computadores Apple, uma versão do Visual Studio pode ser baixada em Visual Studio 2022 for Mac - IDE for macOS (microsoft.com).

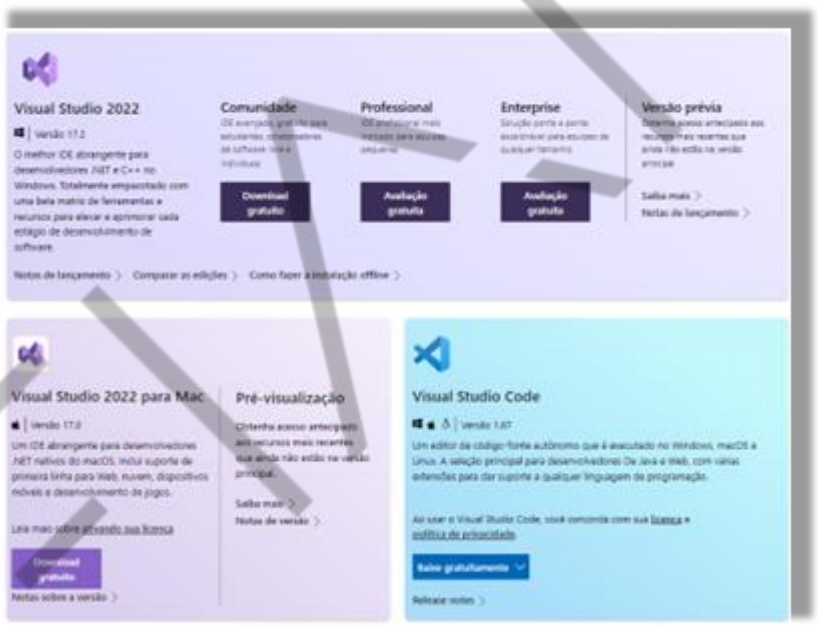

Figura 3 – Download Visual Studio 2022 e documentação
Fonte: Visual Studio (2022)

Após realizar o download e executar o arquivo, irá aparecer a seguinte tela abaixo. Para iniciar, instale as configurações mínimas para desenvolver todas as atividades dos capítulos e poupar recursos do computador. Inclua o pacote de idiomas (inglês), pois os exemplos que irá encontrar em suas pesquisas por muitas das vezes estarão neste idioma.

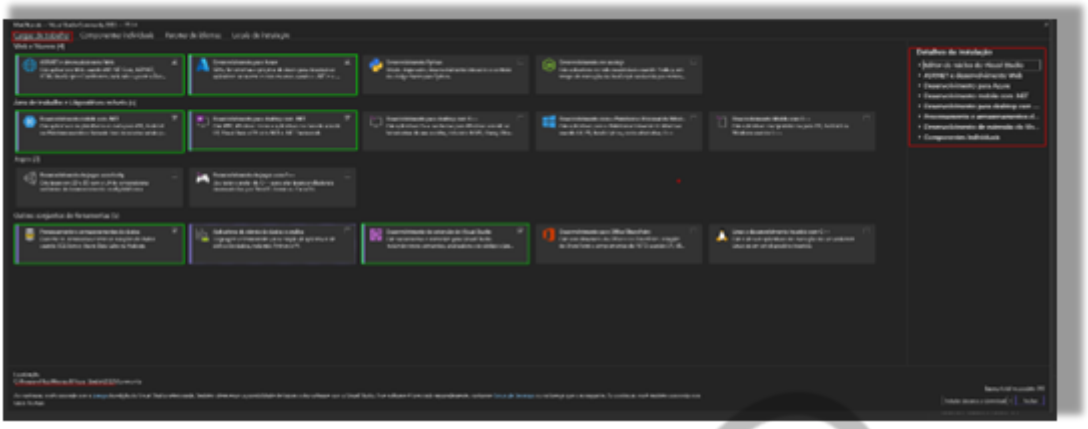

Figura 4 – Pacotes de instalação
Fonte: Visual Studio (2022)

2.4 Criando um projeto

Agora que sabemos um pouco sobre a história do .NET e instalamos o Visual Studio, criaremos o primeiro programa utilizando o Visual Studio e aprofundaremos o 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 a opção Create a **New Project > Console App** e siga os passos abaixo:

- Defina um nome para o projeto (Neste exemplo usaremos FiapHelloWord).
- Defina um diretório no qual será salvo seu projeto.
- Defina um nome para a *Solution* (Por padrão, será o mesmo nome do seu projeto, caso não defina um nome).
- Defina o Framework, neste exemplo utilizaremos o .NET 6.0 (long-term-support).

A sequência abaixo apresenta as telas de criação do projeto:

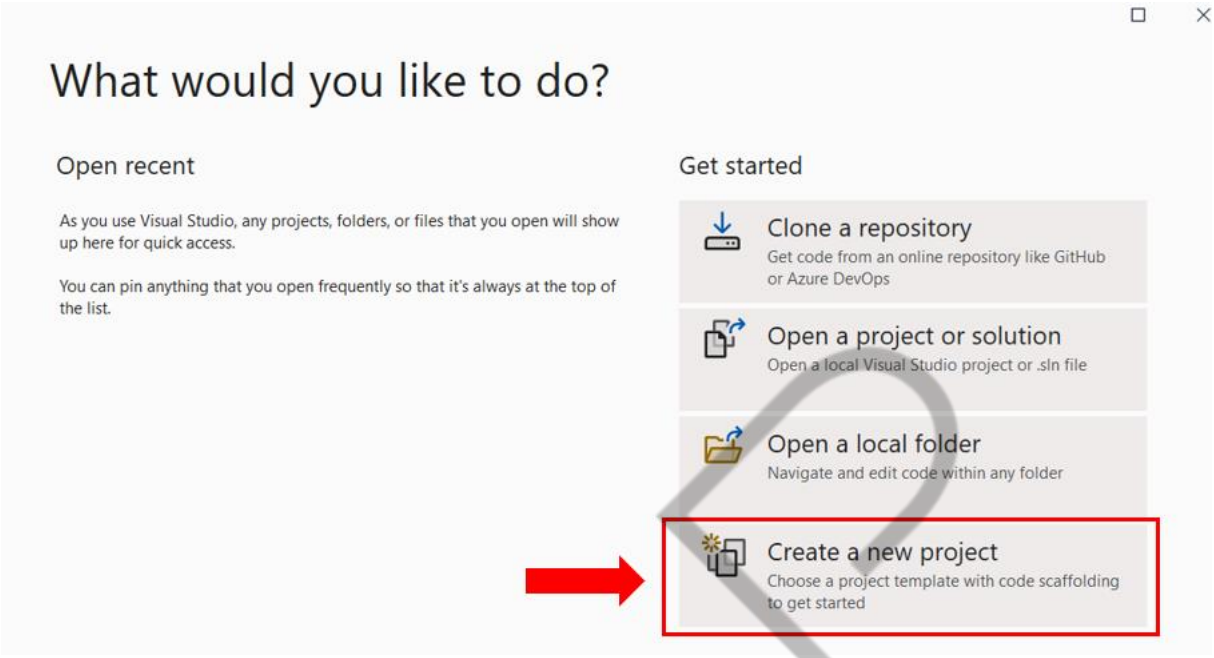

Figura 5 – Tela de criação de projeto
Fonte: Elaborado pelo autor (2021)

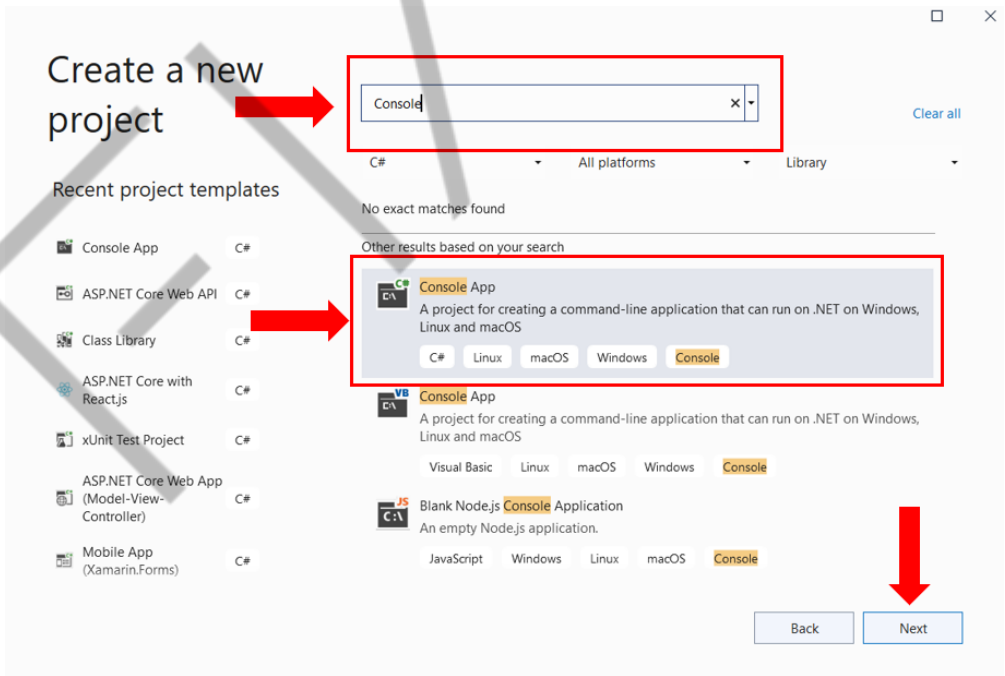

Figura 6 – Selecionando o tipo (scaffold) do projeto
Fonte: Elaborado pelo autor (2022)

Na tela de criação, definimos o nome do projeto, o local no sistema de arquivos (*Location*) e o nome da solução (*Project Name*). Para nosso exemplo, usaremos **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 o conteúdo aberto no editor.

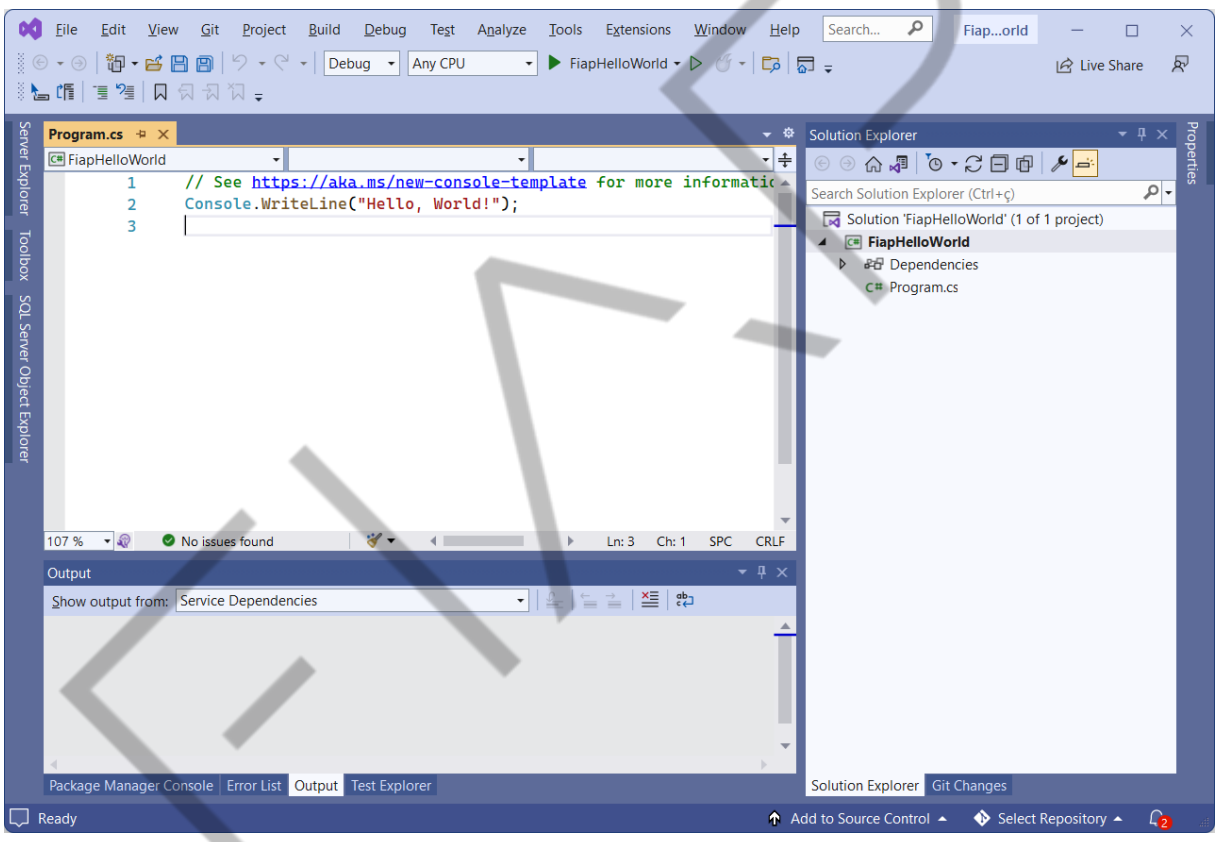

Figura 7 – Projeto FiapHelloWorld
Fonte: Elaborado pelo autor (2021)

Com o projeto aberto, e, por meio do **Solution Explorer (Gerenciador de Soluções)** posicionado do 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 direito em cima do projeto ou da solução, selecione a opção **Open Folder in File Explorer (Abrir pasta no gerenciador de arquivos)**.

2.4.1 Escrevendo 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.

Nesse 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 impressão de uma mensagem na tela e outra linha que mantém a janela para visualização do usuário até que uma tecla seja pressionada.

O código-fonte deverá ficar como no exemplo a seguir:

```
using System;

namespace FiapHelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Fiap – Seja bem vindo a trilha de .NET !");

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

Código-fonte 1 – Primeiro programa no Visual Studio
Fonte: Elaborado pelo autor (2017)

DICA: O código-fonte anterior apresenta o modelo de usados nas versões anteriores a versão .NET 6, para as novas versões apenas o comando Console.WriteLine é necessário.

As duas formas representam o mesmo programa. Ambos são válidos com C# 10.0. Quando você usa a versão mais recente, você só precisa escrever o corpo do método. O compilador sintetiza uma classe com um método e coloca todas as suas

declarações de nível superior nesse método. Você não precisa incluir os outros elementos do programa, o compilador os gera para você.

Você tem duas opções para trabalhar com tutoriais que não foram atualizados para usar modelos .NET 6+:

- Use o novo estilo do programa, adicionando novas instruções de alto nível à medida que você adiciona recursos.
- Converta o novo estilo de programa para o estilo mais antigo, com uma classe e um método *Program Main*.

2.4.2 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.

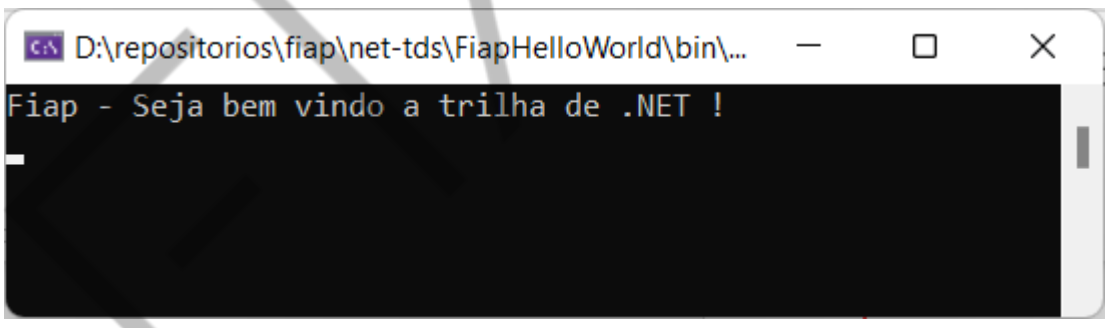

Figura 8 – Janela de execução do aplicativo console
Fonte: Elaborado pelo autor (2018)

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 pressione “Shift + F5”.

A tecla F5 é a forma mais simplificada 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 demais ações podem ser acionadas pelo menu ***Build* (Compilação)** 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. Na Figura “Barra de ferramentas de compilação e execução”, a seguir, é possível visualizar os botões de atalho para compilação e execução:

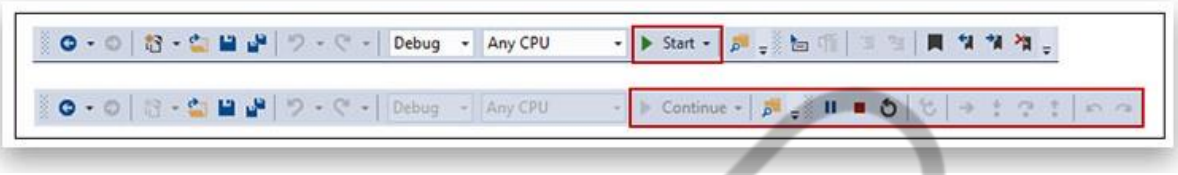

Figura 9 – Barra de ferramentas de compilação e execução
Fonte: Elaborado pelo autor (2018)

2.4.3 Organizando o projeto

Para organizar nossos projetos em C#, precisamos falar de *namespaces*. É uma palavra-chave da linguagem de programação C#. A *namespace* é usada para declarar um escopo que contém um conjunto de objetos relacionados. Você pode usar um *namespace* para organizar elementos de código e criar tipos globalmente exclusivos.

Veja o exemplo na Figura “*Namespace* padrão criado pelo Visual Studio”:

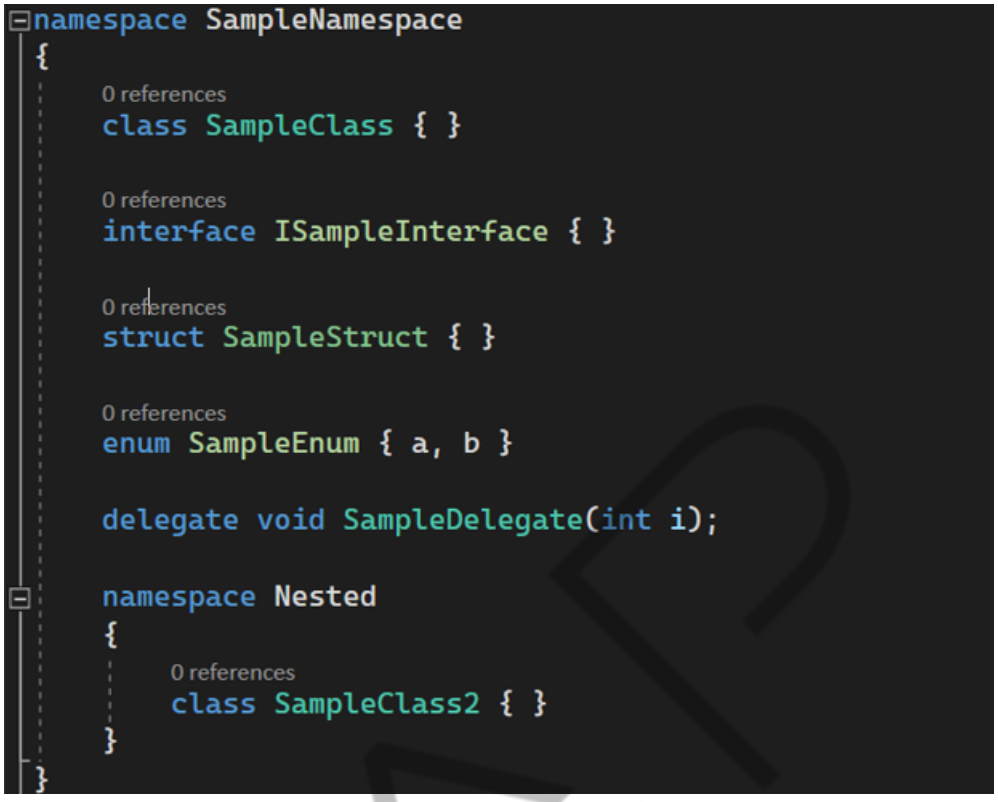

Figura 10 – *Namespace* padrão criado pelo Visual Studio
Fonte: Elaborado pelo autor (2022)

As declarações de *namespace* no escopo do arquivo permitem declarar que todos os tipos em um arquivo estão em um único *namespace*. As declarações de *namespace* no escopo do arquivo estão disponíveis com o C# 10. O exemplo a seguir é semelhante ao anterior, mas usa uma declaração de *namespace* com escopo de arquivo.

```
using System;

namespace SampleFileScopedNamespace;

0 references
class SampleClass { }
```

Figura 11 – Exemplo de um *namespace*
Fonte: Elaborado pelo autor (2018)

O exemplo anterior não inclui um *namespace* aninhado. *Namespaces* com escopo de arquivo não podem incluir declarações de *namespace* adicionais. Você não pode declarar um *namespace* aninhado ou um segundo *namespace* com escopo de arquivo.

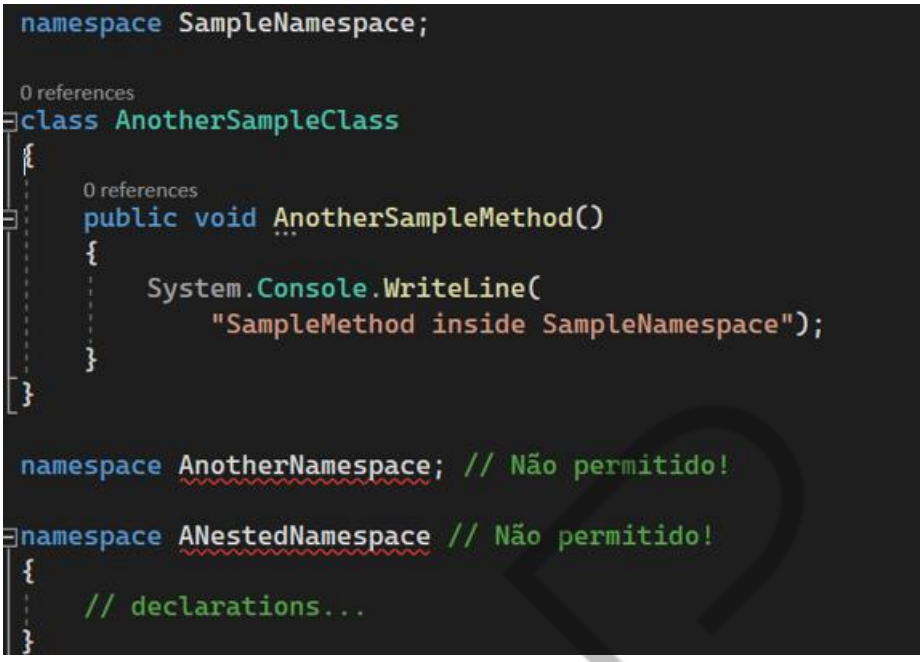

Figura 12 –Exemplos de *namespace*
Fonte: Elaborado pelo autor (2022)

Dentro de um namespace, é possível declarar os seguintes tipos:

- class
- interface
- struct
- enumeração
- delegate
- namespaces aninhados podem ser declarados, exceto em declarações de namespace com escopo de arquivo

O compilador adiciona um *namespace* padrão. Este *namespace* sem nome, às vezes chamado de *namespace* global, está presente em todos os arquivos. Ele contém declarações não incluídas em um *namespace* declarado. Qualquer identificador no *namespace* global está disponível para uso em um *namespace* nomeado.

Os *namespaces* são usados intensamente em programações de C# de duas maneiras. Primeira, o .NET usa *namespaces* para organizar suas muitas classes, da seguinte maneira, seja o código-fonte abaixo:

```
System.Console.WriteLine("Fiap – Seja bem-vindo a trilha de .NET");
```

Código-fonte 2 – Exemplo de uso do *namespace*
Fonte: Elaborado pelo autor (2022)

A palavra ***System*** é um *namespace*, já ***Console*** é uma classe do *namespace* ***System***.

Podemos simplificar esse comando adotando a palavra-chave ***using,*** que pode ser usada para que o nome completo não seja necessário, como no exemplo do código-fonte a seguir:

```
using System;

Console.WriteLine("Fiap – Seja bem-vindo a trilha de .NET");
```

Código-fonte 3 – Exemplo de uso da palavra using
Fonte: Elaborado pelo autor (2022)

Para a criação de um *namespace*, basta clicar com o botão direito no projeto C#, escolher a opção ***Add > New Folder*** e digitar o nome da pasta. Para o nosso exemplo, crie uma pasta com o nome Models.

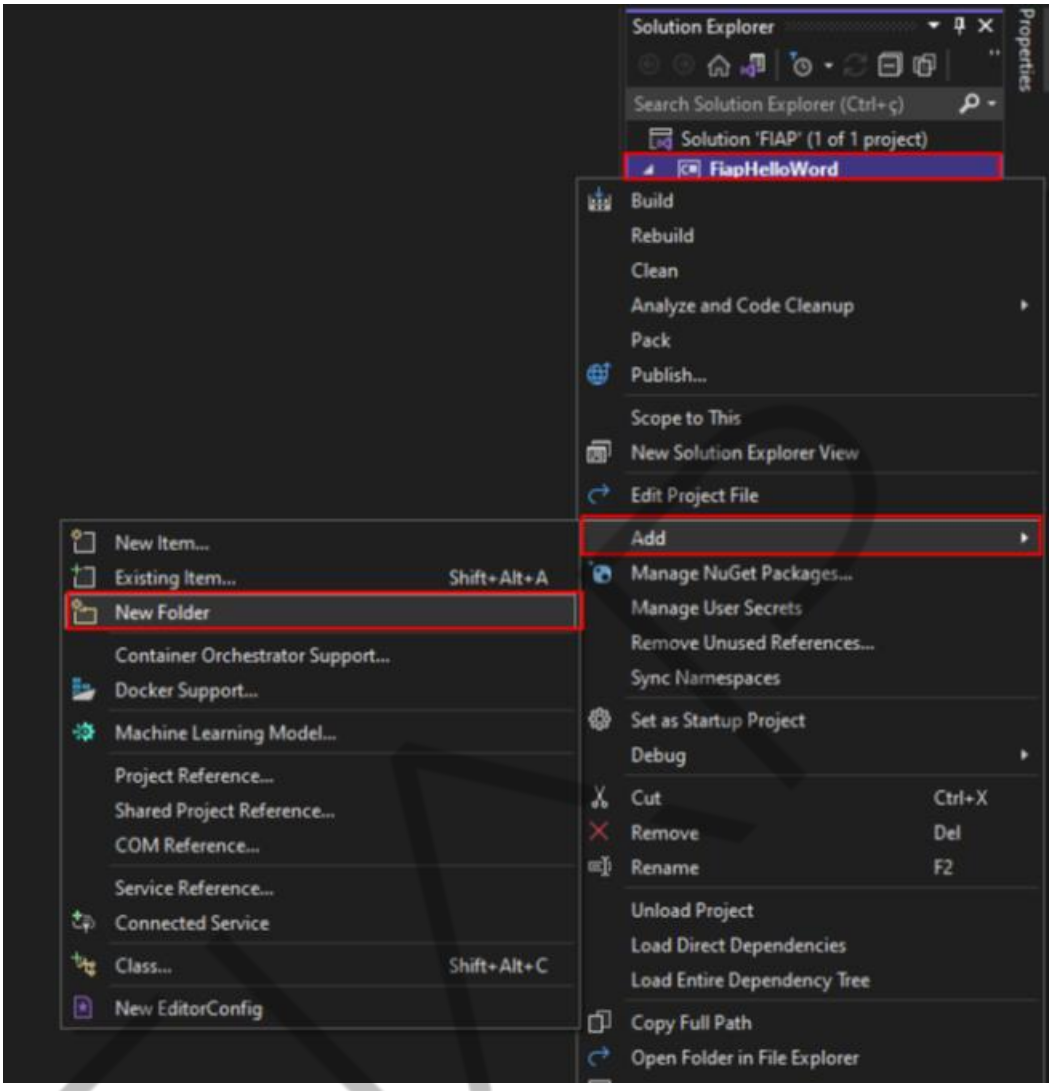

Figura 13 – Criando pasta e *namespace*
Fonte: Elaborado pelo autor (2022)

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 **HelloModel**, como os exemplos abaixo:

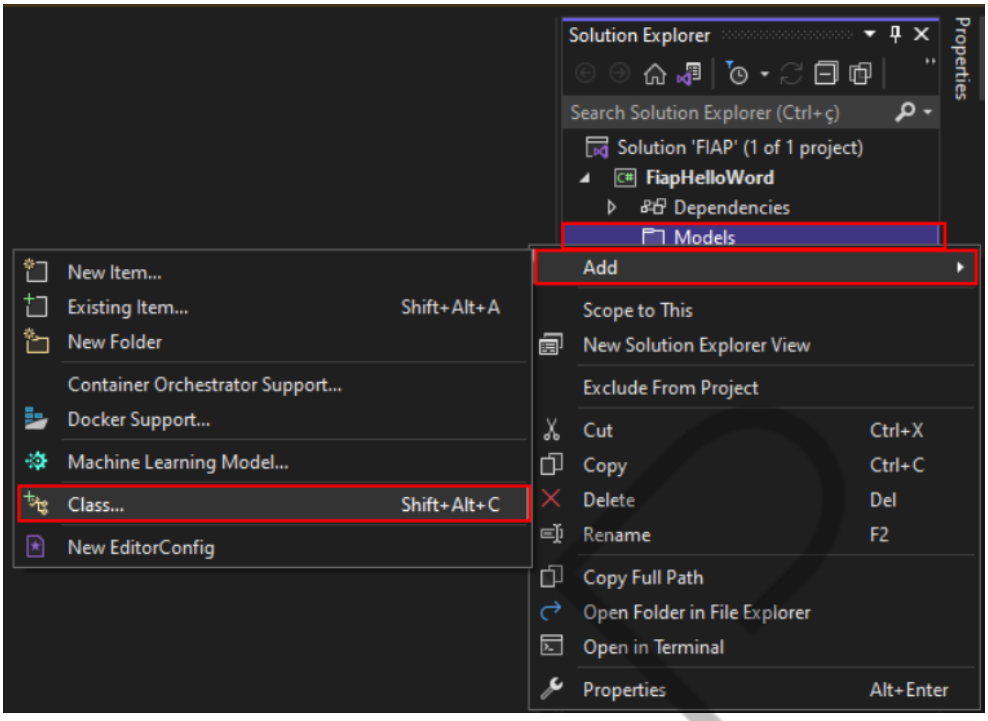

Figura 14 – Adicionando nova classe ao namespace
Fonte: Elaborado pelo autor (2022)

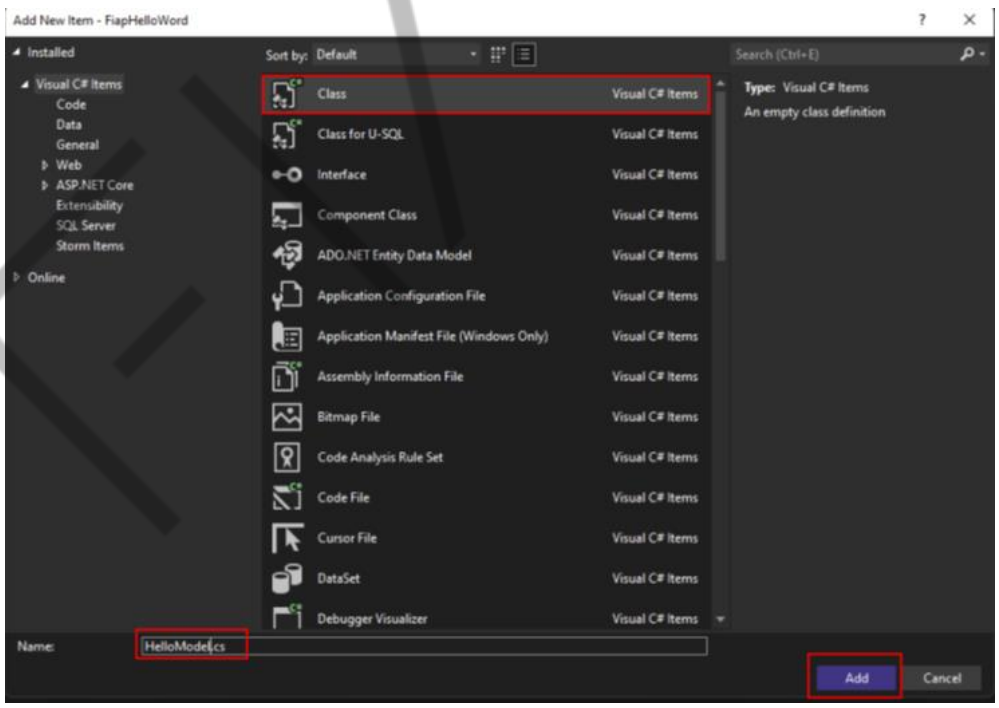

Figura 15 – Selecionando o tipo do artefato Classe
Fonte: Elaborado pelo autor (2022)

Analisando a figura da classe criada abaixo é possível notar que foi definido o *namespace* **FiapHelloWorld.Models** como padrão. E possível definir outro nome para um *namespace* sem mudar o nome da pasta, porém, manteremos os nomes iguais.

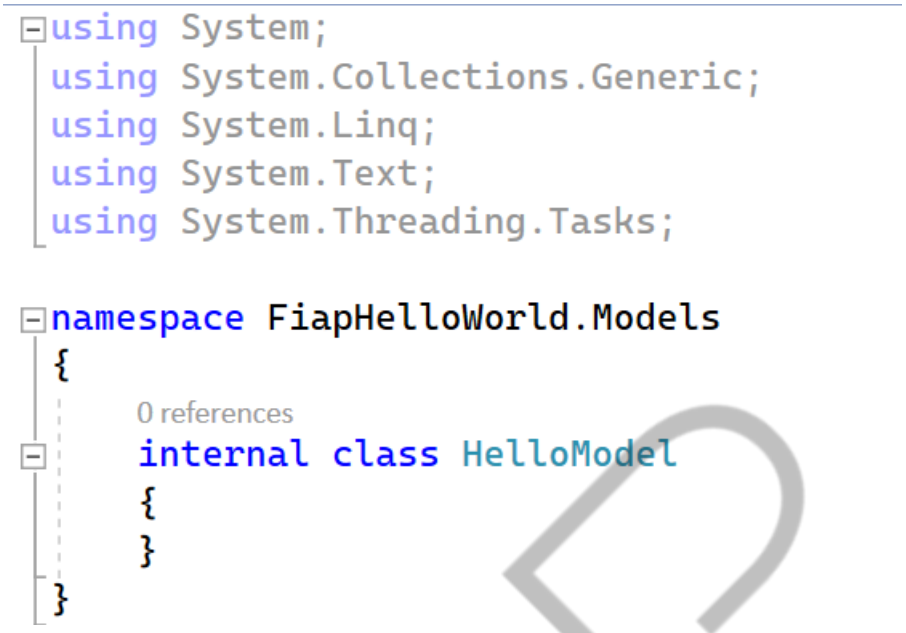

Figura 16 – Exemplo da classe HelloModel
Fonte: Elaborado pelo autor (2022)

Iremos alterar o nosso exemplo e fazer uso da classe de modelo e validar a organização do projeto. Para isso, definiremos uma propriedade de mensagem na classe de modelo, conforme o código-fonte “Classe de exemplo do *namespace* Models”, segue o código abaixo:

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

namespace FiapHelloWorld.Models
{
   internal class HelloModel
   {
      public string Mensangem = "Eu sou uma aluno da Fiap na trilha de .NET";
   }
}
```

Código-fonte 4 – Classe de exemplo do namespace Models
Fonte: Elaborado pelo autor (2022)

Agora vamos alterar a classe **Program.cs** para acessar a classe modelo e imprimir o texto declarado na propriedade **Mensagem**. Veja o código-fonte abaixo:

```
using FiapHelloWorld.Models; // Namespace da classe HelloModel

var helloModel = new HelloModel(); // instancia do objeto HelloModel

Console.WriteLine(helloModel.Mensangem); // imprimindo conteúdo da propriedade Mensagem

Console.ReadKey();
```

Código-fonte 5 – Usando uma classe de outro namespace
Fonte: Elaborado pelo autor (2022)

Basta executar o programa e verificar o resultado a impressão do conteúdo semelhante a tela a seguir:

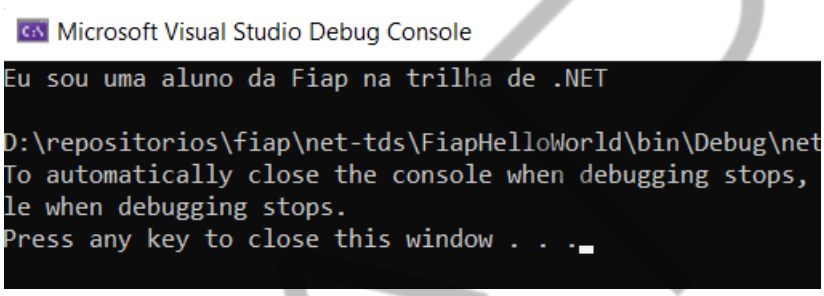

Figura 17 – Resultado da execução do teste de namespace
Fonte: Elaborado pelo autor (2022)

2.5 Debug

Até este ponto, conseguimos executar uma aplicação usando o Visual Studio e a linguagem C#. Agora, precisamos 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.

Iniciaremos 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 com um clique na margem esquerda da janela de editor, podemos adicionar um ponto de interrupção (*breakpoint*). A Figura “Ponto de interrupção (*breakpoint*)” apresenta a linha selecionada e o ponto de interrupção criado.

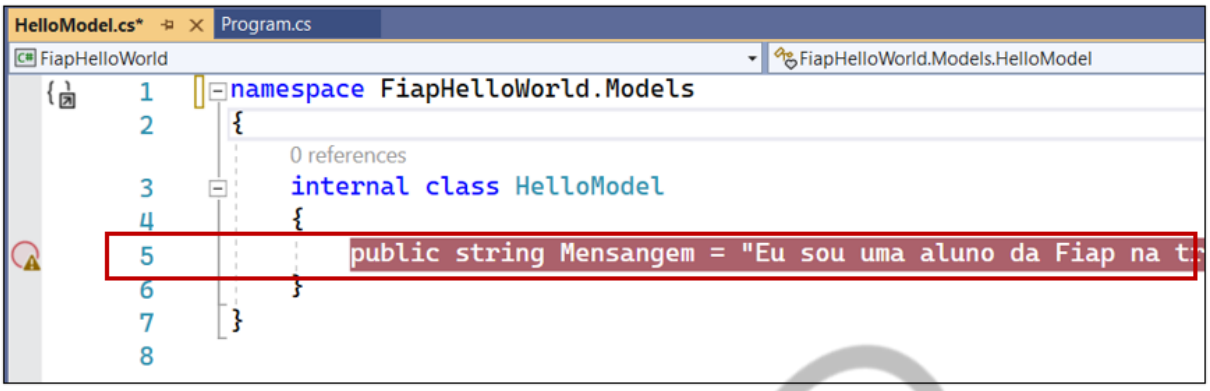

Figura 18 – Ponto de interrupção (*breakpoint*)
Fonte: Elaborado pelo autor (2022)

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 no entendimento do programa podem ser tomadas. A seguir, veremos detalhes das ações mais comuns de debug.

2.5.1 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**.

Na 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” apresenta a ferramenta immediate, uma linha de comando para alteração do conteúdo de uma propriedade e o resultado da alteração.

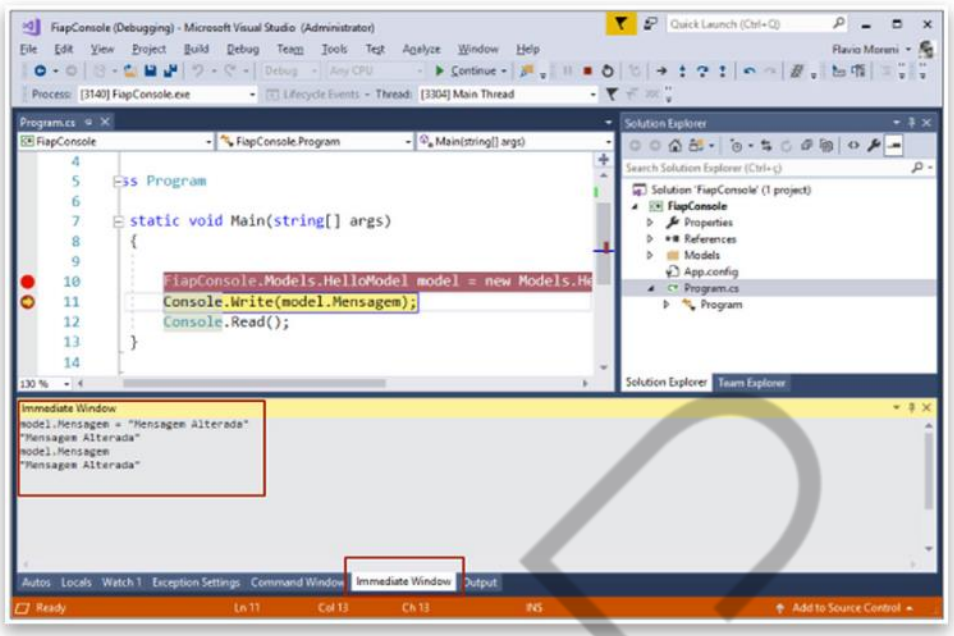

Figura 19 – Janela Immediate Window
Fonte: Elaborado pelo autor (2022)

2.5.2 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 pelas teclas **Shift + F9**.

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

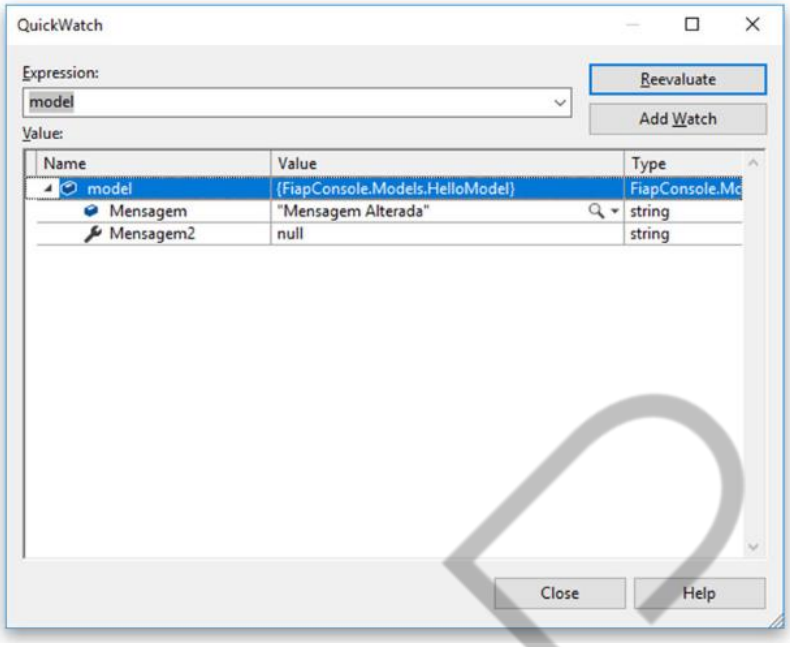

Figura 20 – Janela *Quick Watch*
Fonte: Elaborado pelo autor (2018)

2.5.3 Navegando pelo código

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

- *Step-Into* (F11) executa 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) avança a execução dentro da mesma estrutura de código.
- *Step Backward* (Alt + ´) retrocede a execução para a linha anterior.
- *Continue* (F5) executa a aplicação até o próximo *breakpoint.*
- *Step Out* (Shift + F11) retorna ao ponto de debug que originou a chamada.

As ações de *Debug* podem ser usadas por meio das teclas de atalho, na opção *Debug* no menu superior ou nos botões da barra de ferramenta. A Figura “Barra de ferramentas para *debug*” apresenta as opções de *debug* na barra de ferramentas.

Figura 21 – Barra de ferramentas para *debug*
Fonte: Elaborado pelo autor (2018)

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

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

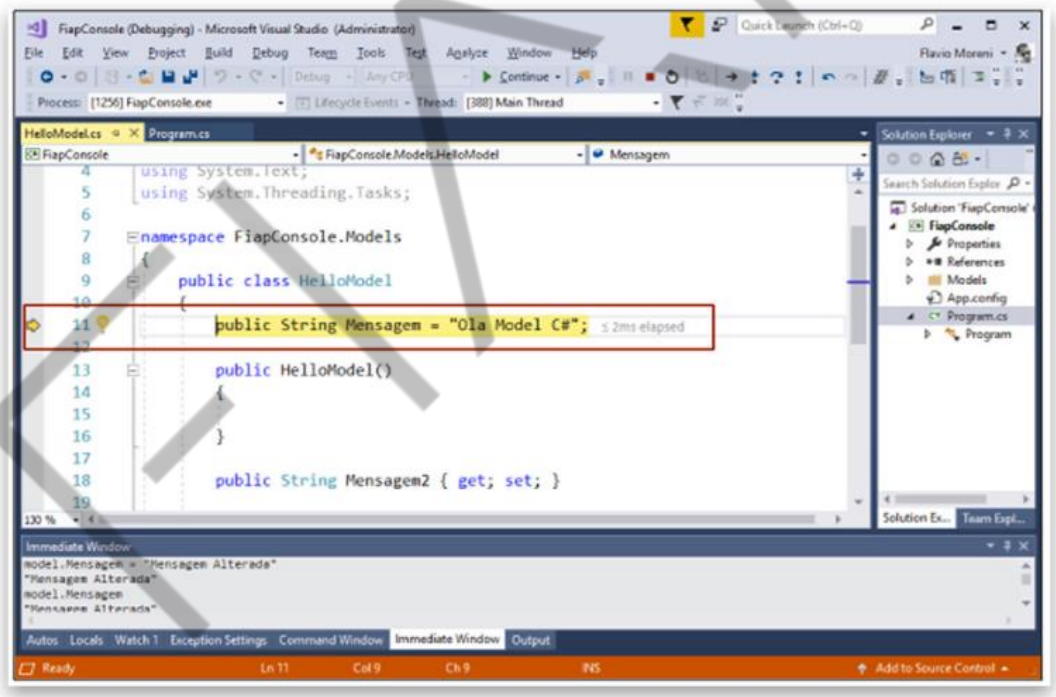

Figura 22 – Depurando passo a passo
Fonte: Elaborado pelo autor (2018)

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

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

2.6 Atalhos

Assim como as demais *ides*, o Visual Studio fornece diversas teclas de atalhos para facilitar o dia a dia do desenvolvedor e aumentar a produtividade na construção de código. O Quadro “Atalhos do Visual Studio”, abaixo, apresenta 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 opção de *SmartTags* para correções rápidas de código ou criação rápida de código. É usada 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 – Atalhos do Visual Studio
Fonte: FIAP (2016)

CONCLUSÃO

O conteúdo deste capítulo teve como objetivo apresentar a plataforma .NET, o resumo da história dessa tecnologia. Foram abordados de forma introdutória o .NET Framework, as linguagens suportadas pelo framework, os componentes e a criação da linguagem C Sharp (C#).

Foi apresentada a ferramenta Microsoft Visual Studio, a forma de obtenção, além das linguagens suportadas e os tipos de projetos que podem ser criados. Por fim, foi realizada a criação de um projeto simples para familiarização com a IDE, a organização, a compilação e a 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. **Introdução ao .NET Framework**. [s.d.]. Disponível em: https://msdn.microsoft.com/pt-br/library/hh425099(v=vs.110).aspx. Acesso em: 13 jan. 2021.

MICROSOFT. **Atalhos de teclado padrão no Visual Studio**. [s.d.]. Disponível em: https://msdn.microsoft.com/pt-br/library/da5kh0wa.aspx. Acesso em: 13 jan. 2021.

MICROSOFT. **Namespace (Referência de C#)**. [s.d.]. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/namespace. Acesso em: 13 jan. 2021.

GLOSSÁRIO

VS	*Visual Studio*