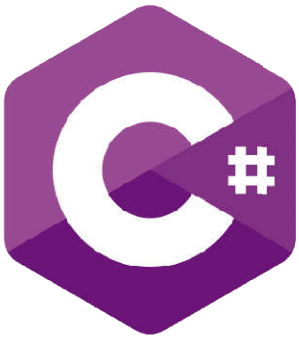
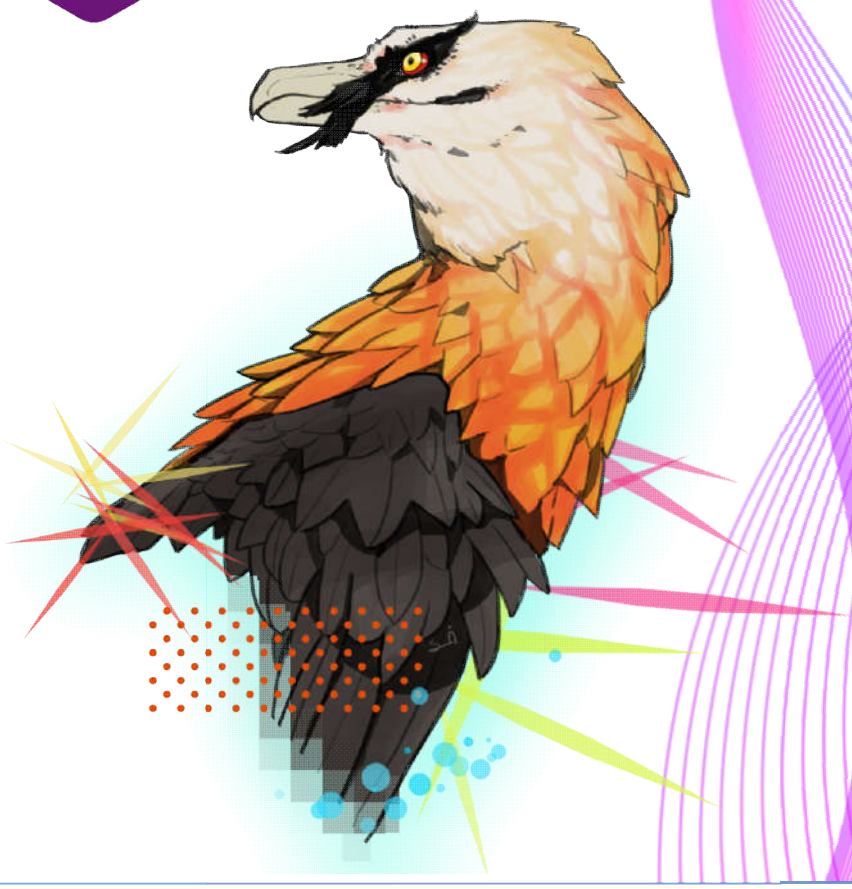
سی شارپ

در متن باز

پدرام رحیمی



in open source



فهرست

آشنایی با سی شارپ

زبان C# در ابتدا به نام COOL مخفف **C** **O**bject **O**riented **L**anguage شناخته شده که علاوه بر ساده تر کردن زبان سی، امکانات شی گرایی را به زبان آن اضافه کرده بود تا برنامه نویسان بتوانند لایه های مختلفی از کدها را با اسامی مختلف در برنامه ها بنویسند و فهم کدها را با طبقه بندی آنها آسان تر کرده و ساختارهای امنیتی اصولی تری داشته باشند. در واقع این زبان سال ۲۰۰۰ میلادی از دل کتابخانه های کدی به نام Net framework. (با تلفظ دات نت فریم ورک) بیرون آمده است. بیشتر کدهای این کتابخانه ی کد نیز با زبان C++ (سی پلاس پلاس) نوشته شده اند.

فریموُرک در واقع کتابخانه ای پیشرفته از کدهای یک زبان است که دیگر بی نیاز از خود زبان می توان آن را به کار برد و البته مثل کتابخانه های ساده امکان همراهی با هر کتابخانه ی مجزایی هم در آن دیگر نیست و خودش را میتوان یک زبان مستقل دانست.

شرکت مایکروسافت ارائه کننده ی این زبان مستقل در نهایت اواخر سال ۲۰۱۰ انحصار خودش را از روی آن برداشت و اولین نسخه ی متن باز یا Open source این کتابخانه را با نام دات نت Core عرضه کرد که بر روی سیستم عامل های دیگر غیر از ویندوز مثل لینوکس نیز قابل استفاده باشد. اگر چه این کار باعث شد طرفداران زیادی پیدا کند اما توسعه ی این کتابخانه ها توسط خود مایکروسافت ادامه پیدا کرد تا جایی که نسخه ی دات نت Standard بعد از اولین نسخه ی اوپن سورس در سال ۲۰۲۰ ارائه شد. نسخه ای که کمک می کند علاوه بر قابلیت اجرای چندسکویی یا Cross platform که هدف آن اجرا روی سیستم عامل های مختلف است، تغییرات جدید باعث شد برای تمام منظورها از برنامه نویسی موبایل، دسکتاپ و حتا وب، فقط یک بار کدنویسی کافی باشد و به نوشتن کدهای مخصوص برای هر ابزار یا فرم اجرا دیگر نیازی نداشته باشیم.

یکی از پرکاربردترین نمونه های کدنویسی با این زبان، از طریق تکنولوژی قدیمی تر صفحات پویای وب یا **A**ctive **S**erver **P**age است که به اختصار ASP نامیده میشد و از زمان همراه شدنش با سی شارپ آن را به نام ASP.net می شناسیم. این کتابخانه ی تحت وب بهترین گزینه برای برنامه نویسی اوپن سورس با سی شارپ در همه جاست. چون در واقع علاوه بر دارا بودن بیشترین امکانات این زبان، می توان روی هر مرورگر اینترنتی یا Browser صرف نظر از نوع

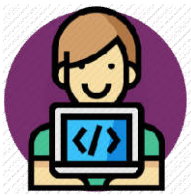
دستگاه یا سیستم عامل، اجرای آن را دید. حال آنکه یک برنامه ی تحت دسکتاپ (مثلاً ویندوزی) یا تحت موبایل چنین خاصیتی را ندارد و فقط در محیط ویژه ی خودشان اجرا می شوند.

اما شیطنت بزرگ مایکروسافت در واقع این بوده است که همچنان عده ی زیادی از برنامه نویسان را به استفاده از محیط نگارش ویژه ی سی شارپ با نام Visual studio وابسته نگه داشته است. چرا که کار با این محیط چنان آسان بوده است که برنامه نویس حتا برای نسخه های تحت وب نیازی به یادگیری HTML یعنی زبان طراحی اصلی و کلاسیک صفحات وب را نداشته است. این ابزار اگر چه توسعه را راحت میکند اما در اجرا با مشکلات بسیاری همراه بوده و علاوه بر پولی بودن نسخه ی کاملش (محیط کدنویسی غیر اوپن سورس برای یک زبان اوپن سورس!) نیاز به سخت افزار قوی و حجم بالایی از منابع سیستمی برای اجراست.

به همین دلیل در اینجا با روش کار به صورت اوپن سورس و با ابزارهای رایگان و مطرح جهانی آشنا می شویم. روشی که شاید بتواند شانسی را برای استفاده ی صحیح و غیرانحصاری واقعی برای این زبان پُر امکانات فراهم کند. در این روش متفاوت با دوره های رایج آموزش این زبان، یادگیری اصولی کار با سی شارپ و کتابخانه هایش، باعث می شود که استفاده از روش کلاسیک یعنی کار در ویژوال استودیو هم ساده و ممکن گردد. در حالی که برعکس این قضیه ممکن نیست. یعنی کسی که به محیط انحصاری مایکروسافت عادت دارد، برای کار و یادگیری روش بهره برداری از طریق خط فرمان، در سیستم عامل های دیگر، برایش مشکل خواهد بود.

تمام کدهای این کتاب تست شده و درست هستند اما برای فهم آنها لازم است یک بار خودتان تایپ و اجرایشان کنید و از خواندن و مطالعه بدون تست پرهیز کنید چون برنامه نویسی علاوه بر دانش یک مهارت است.

برای ارتباط با نویسنده ی این کتاب و دریافت نمونه کدهای بیشتر، به کانال تلگرامی "کدنویس یکروزه" به نشانی **@CODINGINADAY** مراجعه کنید.



راه اندازی سریع

راه اندازی مترجم (Compiler) برای ترجمه ی کدهای سی شارپ به زبان ماشین و اجرایش روی کامپیوتر، در آخرین نسخه ی آن برای تمام سیستم عامل های معروف، بسیار ساده است. علاوه بر این که اوپن سورس بودن (انحصاری نبودن) باعث می شود که برنامه نویسان مختلفی از سراسر دنیا از این به بعد روی پیشرفت آن بتوانند کار کنند، در اصل کمک می کند که زبان سی شارپ مانند نمونه های دیگر محصولات مایکروسافت یعنی Visual BASIC استفاده اش منسوخ نشود. کافیست عبارت **dotnet SDK** را در جستجوگر گوگل سرچ کنید. عبارت SDK مخفف **S**ystem **D**evelopment **K**it یا بسته ی برنامه سازی سیستم است که می توان آن را از نشانی زیر نیز مستقیم دریافت کرد:

https://dotnet.microsoft.com

روی عبارت **Download** که بزنید نسخه ی منتشر شده ی جاری با برچسب **Current** و نسخه ی تست شده به صورت طولانی مدت با برچسب **LTS** که مخفف **L**ong **t**ime **s**upport است، آماده ی دریافت خواهد بود. بعد از اینکه نسخه ی مناسب سیستم عامل خود را انتخاب کردید، مثلاً برای ویندوز ۶۴ بیتی فایلی با نامی شبیه **sdk-5.0.100-windows-x64installer.exe** برای شما دانلود شده که بعد از نصب در محیط خط فرمان قابل آزمایش است. برای لینوکس هم یک سلسله دستورات قرار داده شده اند که کافیست از سایت مذکور کپی و در خط فرمان یا ترمینال لینوکس وارد کنید تا نصب به صورت خودکار انجام شود. نصب که به پایان رسید، خط فرمان یا Command prompt (در لینوکس Terminal) را باز کنید. دستور دات نت را به صورت تنها وارد کنید:

dotnet

حالا عبارات زیر را خواهید دید:

```
Usage:  dotnet  [options]

Options:
-  h| --help          Display help.
--  info               Display .NET information.
--  list -sdks         Display the installed SDKs.
```

-- list -runtimes Display the installed runtimes.

این عبارات توضیح می دهند که فرم کلی استفاده از دستور dotnet به چه شکل است. برای این مرحله فقط از آپشن **--help** استفاده کنید:

dotnet --help

.NET SDK 5.0.100
Usage: dotnet [runtime-options] [path-to-application] [arguments]

sdk-options:
- h|--help Show command line help.
-- info Display .NET information.
-- version Display .NET SDK version in use.

SDK commands:

add	Add a package or reference to a .NET project.
build	Build a .NET project.
clean	Clean build outputs of a .NET project.
help	Show command line help.
new	Create a new .NET project or file.
nuget	Provides additional NuGet commands.
pack	Create a NuGet package.

..

و از بین دستوراتی که تحت عنوان commands معرفی شده اند، **new** را انتخاب می کنیم:

dotnet new

Templates	Short Name	Language	Tags
Console Application	console	[C#], F#, VB	Common/Console
Class library	classlib	[C#], F#, VB	Common/Library
WPF Application	wpf	[C#], VB	Common/WPF
Windows Forms App	winforms	[C#], VB	Common/WinForms

...

همانطور که روشن شد، با دستور زیر می توانیم اولین برنامه ی سی شارپ را حالا در هر مسیری که دوست داریم ایجاد کنیم. ترجیحاً درون یک پوشه یا مسیر جدید و مشخص آن را اجرا کنید، مثلاً درون پوشه ای روی دسکتاپ:

```
dotnet   new   console
```

و خواهید دید که پروژه ی جدید ما در مسیری که تعیین کرده ایم ساخته می شود. این نمونه ی اولیه ای است که در هر زبان برنامه نویسی با نام Hello world می شناسیم و به عنوان ساده ترین خروجی هر زبان، یک متن ساده را مثلا سلام بر دنیا باید برگرداند. فعلاً صرف نظر از اینکه کدهای ایجاد شده به چه معنایی هستند از میان فایل های ایجاد شده، فایل **Program.cs** را با یک ویرایشگر ساده ی متن مثل Notepad باز کنید:

```
using System;
namespace Desktop
{
   class Program
   {
         static void Main(string[ ] args)
         {
             Console.WriteLine("Hello World!”);
          }
    }
}
```

تنها به بلوک یا بخشی که بین **{ }** قرار دارد و بالای آن عبارت **Main** دیده میشود، در این فایل توجه کنید. داخل این بلوک دستور ساده ای به نام **Console.WriteLine** خطی را در کنسول یا همان خط فرمان چاپ می کند. عبارت بین دبل کوتیشن همان جمله ی معروف است که می

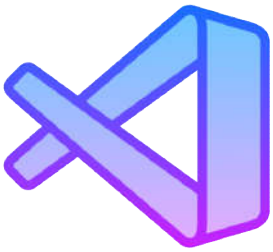
توانید به جایش هر عبارت دیگری قرار داده و دوباره Save کنید. مثلاً Hello friends! را بنویسید. فایل را دوباره ذخیره کنید. در همان خط فرمان و مسیر پروژه دستور زیر را برای اجرا وارد کنید:

```
dotnet run
```

خواهید دید که بلافاصله بعد از این دستور عبارت Hello friends! روی صفحه و داخل کامندلاین نوشته می شود. این **ساده ترین نوع و نحوه ی اجرای برنامه ی سی شارپ** بود. برنامه ای که فقط یک عبارت ساده را به عنوان خروجی به نمایش می گذارد. روشن است که چنین چیزی هدف یک برنامه نویس نمی تواند باشد اما امکان اجرای صحیح سی شارپ و تست محیط کنسول یا خط فرمان را فراهم می کند.

به علت وسعت امکانات این زبان و زیاد بودن دستورات، همچنین حساس بودن به حروف بزرگ و کوچک، به خاطر سپردن کدها مشکل می شود. به همین دلیل نگارش کدهای سی شارپ با یک نوتپد یا هر ادیتور ساده کار صحیحی نیست چون اول اینکه به خاطر سپردن املای دستورات روش اشتباهی است و برنامه نویس باید فقط وقت خود را به یادگیری امکانات یک زبان اختصاص دهد و از طرف دیگر هر لحظه امکان بروز خطاهای نگارشی وجود دارد. به همین دلیل وجود یک مکان ویژه ی تایپ دستورات الزامی است. محیط یکپارچه ی تایپ و اجرای یک زبان برنامه نویسی را IDE می نامند که مخفف Integrated **D**evelopment **E**nvironment است.

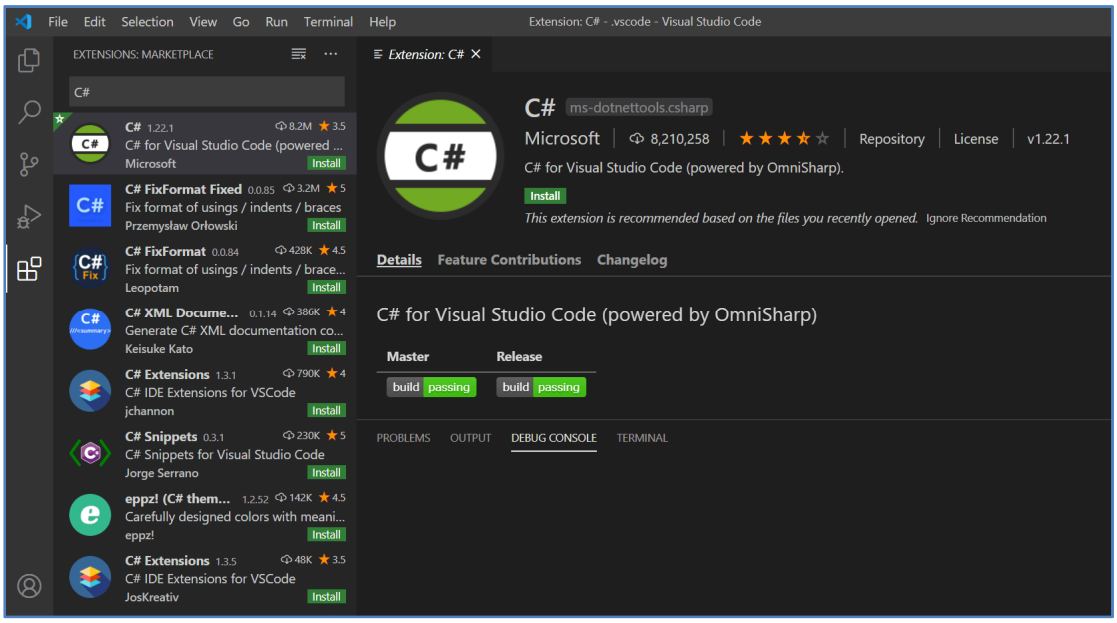
محیط کدنویسی



برای اجرای سی شارپ بعد از سالیان متمادی دیگر محدود به ویرایشگر حجیم و غیر رایگان خود مایکروسافت یعنی Visual studio نیستید. علاوه بر پروژه های به نسبت جالب و موفق و رایگانی که در این زمینه تولید شده اند، نیز نمونه ی پولی JetBrain Rider یکی از بهترین هاست. اما در اینجا برای شروع کدنویسی، نرم افزار رایگان و اوپن سورس خود مایکروسافت را به شما پیشنهاد می دهیم:

https://code.visualstudio.com

این برنامه که به نام VScode (با تلفظ وی اس کد) در بین برنامه نویسان شناخته می شود، یک محیط ایده آل است که بر روی سیستم عامل لینوکس نیز قابل نصب بوده و درست مانند نسخه ی ویندوزی خود عمل می کند. در واقع این محیط محدود به زبان برنامه نویسی خاصی نیست اما برای اینکه امکانات ویژه ی هر زبانی مثل خودتکمیلی کدها (Intellisense) به آن اضافه شود کافیست بعد از نصب خود برنامه و اجرای آن، افزونه ی مربوط به آن زبان را هم از بخش Extensions دریافت و نصب کنید. حتا به محض اینکه یک فایل سی شارپ با پسوند cs. را توسط این ویرایشگر کد باز کنید، ممکن است به طور خودکار افزونه ی مناسب را به شما پیشنهاد دهد. البته تعداد این افزونه ها زیاد هستند چون مثل هر محیط اوپن سورس دیگر، برنامه نویسان زیادی اقدام به تهیه ی این افزونه ها کرده اند.



به راحتی با انتخاب نمونه های مهمتر افزونه ها که از روی رتبه بندی و آمار نصب شناخته می شوند، می توان مناسب ترین را نصب کرد. انتخاب گزینه ی Install در کنار نام هر افزونه باعث نصب کامل آنها می شود.

بعد از این محیط ویرایشگر کد شما کامل و آماده است. کافی است از منوی فایل این ویرایشگر گزینه ی Open folder را انتخاب و پوشه ی پروژه ی ساخته شده ی خود را به آن بدهید تا این مسیر در ویرایشگر کد شما باز شود. از طرفی اگر در کامندپرامپت و مسیر پروژه یعنی همانجایی که دستور run را اجرا کردید هستید، کافیست دستور زیر را وارد کنید:

code .

به نقطه ی بعد از دستور که با یک فاصله قرار داده می شود، توجه داشته باشید. در اینصورت وی اس کد در مسیر پروژه ی شما باز می شود. فایل های پروژه نیز در پنل سمت چپ دیده می شوند. با انتخاب هر فایل امکان ویرایش و ذخیره ی مجدد آن با CTRL+S وجود دارد.

اجرای حرفه ای

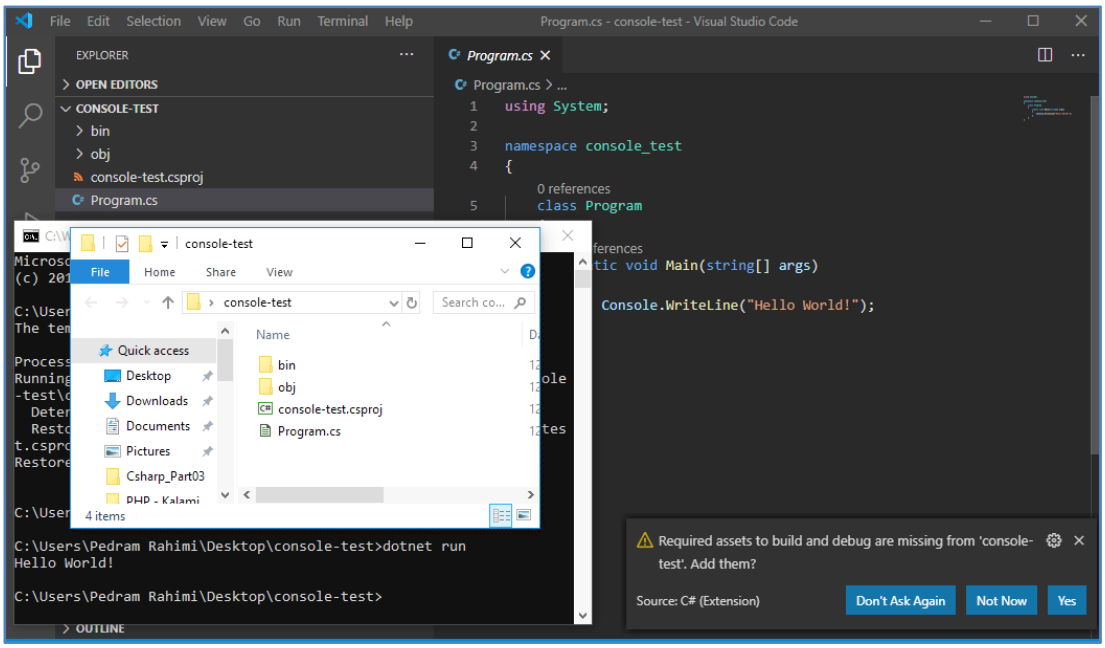
برنامه ی "سلام دنیا" را که ساخته اید در محیط ویرایشگر باز کنید. شاید متوجه شده اید که اولین اجرای برنامه با dotnet run قدری طولانی بود. دلیلش زمینه سازی هایی قبل از اجرای دستورات بوده است. به این کار یا بازسازی اولیه Build یا بیلد کردن گفته می شود. در واقع برخلاف خیلی از زبانهای دیگر مثل PHP که اجرا کننده ی آنها یک مُفسّر (interpreter) است و کد آن زبان را درجا به زبان ماشین و قابل اجرا برای کامپیوتر تبدیل می کند، در سی شارپ مُترجم یا همان Compiler ابتدا کدهای شما را به یک زبان قابل اجرا و بومی شده (Native) توسط سیستم عامل تبدیل کرده و بعد این نسخه ی خیلی سریعتر از یک زبان مُفسری، توسط سیستم عامل اجرا می شود. همین موضوع باعث می شود که اگر دفعات بعد تغییری روی کدها نداده باشیم، اجرای برنامه سریعتر و از روی کدهای بیلد شده انجام شود نه کدهای سی شارپی که به زبان انسان نزدیک تر است. و این فرق مهم مُفسر و مترجم در زبانهای برنامه نویسی است.

همینطور **اگر در سیستم عامل ویندوز باشید و بخواهید یک نسخه ی قابل اجرا از برنامه ی خود را به کسی بدهید و او بدون دیدن کدها و بی نیاز از نصب سی شارپ بخواهد آن را اجرا کند، کافیست پوشه ی bin از پروژه ی خود را فقط بفرستید و از او بخواهید فایلی را که پسوند exe. دارد اجرا کند.**

یک بار دیگر مراحل ساخت و اجرای یک پروژه را با ویرایشگر وی اس کد این بار مرور می کنیم:

- روی دسکتاپ ویندوز کلیک راست کرده با **New -> Folder** پوشه ای ساخته و نام آن را مثلاً **console-test** گذاشته و پوشه را باز کنید.
- در نوار آدرس پنجره ی باز شده ی پوشه، آدرس را پاک کرده و تایپ کنید **cmd** و کلید Enter را بزنید. در این صورت خط فرمان در مسیر این پوشه باز می شود. اما در لینوکسی مانند Ubuntu کافیست یک ترمینال در پنجره ی باز شده ی پوشه اجرا کنید.

- حالا دستور **dotnet new console** را در همین مسیر وارد کنید و صبر کنید تا فایل های پروژه ساخته شوند.
- دستور **code .** را هم با همین فرمت یعنی یک فاصله و نقطه بعد از آن وارد کنید تا VScode نیز در مسیر پروژه باز شود.
- دستور **dotnet run** را یک بار اجرا کنید و صبر کنید تا عبارت Hello world! نمایش داده شود.



اگر پیامی در پایین VScode نمایش داده شده که می خواهد فایلهای لازم برای مدیریت برنامه ی سی شارپی شما را دانلود کند، آن را تأیید کنید:

Required assets to build and debug are missing from 'console-test'. Add them?

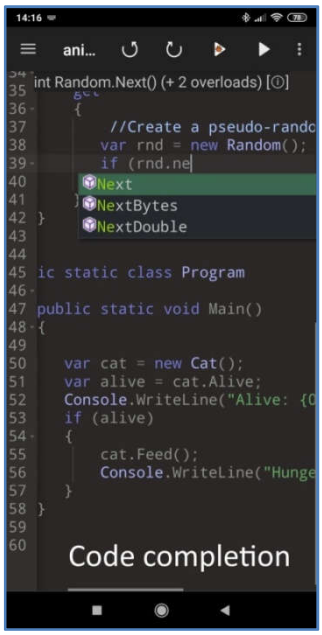
حالا هم برنامه اجرا شده و هم فایلهای آن برای ویرایش و کدنویسی های بعدی آماده هستند. از پنل سمت چپ در وی اس کد روی Program.cs کلیک کنید تا در پنل سمت راست کدهای آن به نمایش در آیند:

```
using System;
namespace console_test
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.ReadKey();
        }
    }
}
```

به خط کد جدیدی که اضافه کرده ایم توجه کنید. دستور **Console.ReadKey** بعد از چاپ Hello World منتظر فشردن یک کلید باقی می ماند تا برنامه را بعد از آن پایان دهد. اگر این خط کد نباشد و شما نسخه ی exe. را اجرا کنید متوجه خواهید شد که برنامه بلافاصله بعد از اجرا، پنجره ی کنسول را می بندد و شما متوجه ی اجرای آن نخواهید شد. پس تا این لحظه دو دستور یکی برای نمایش پیام و دیگری انتظار برای فشردن کلید را دیدید.

نکته: اجرای سی شارپ روی موبایل یا در کافی نت



اگر چه شرکت مایکروسافت به طور رسمی هنوز نسخه ای مخصوص آندروید ارائه نکرده، ولی ابزارهای زیادی برای اجرای کدهای سی شارپ در موبایل وجود دارند.

یکی از بهترین ها را که به صورت آفلاین و بدون اینترنت می توان روی موبایل استفاده کرد در Google play با نام **com.radinc.csharpshell** جستجو کنید.

اما اگر حتا نخواستید که چیزی را هم نصب کنید، نمونه های آنلاین هم وجود دارند که با یک سرچ ساده می توان آنها را به عنوان کمپایلر سی شارپ پیدا کرد. نمونه هایی مثل:

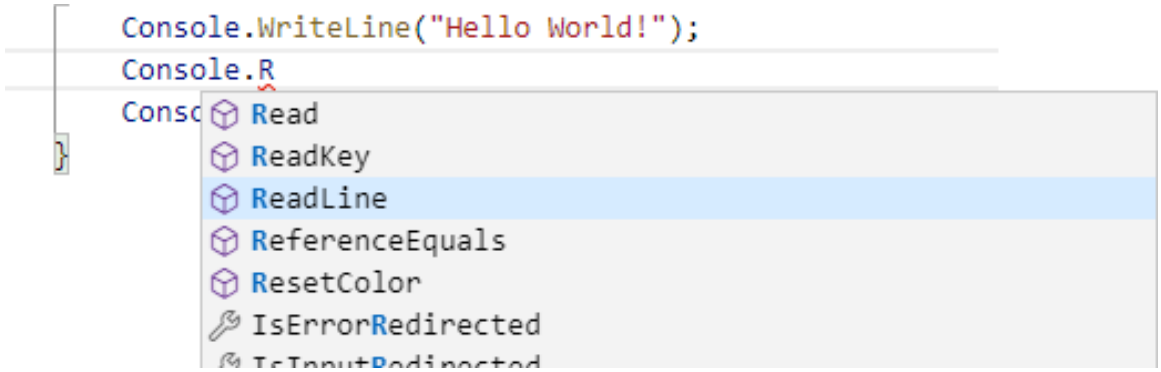
jdoodle.com dotnetfiddle.net rextester.com

شروع کاربردی

حالا نوبت نوشتن اولین برنامه ی کاربردی است. همانطور که تا اینجا متوجه شده اید:

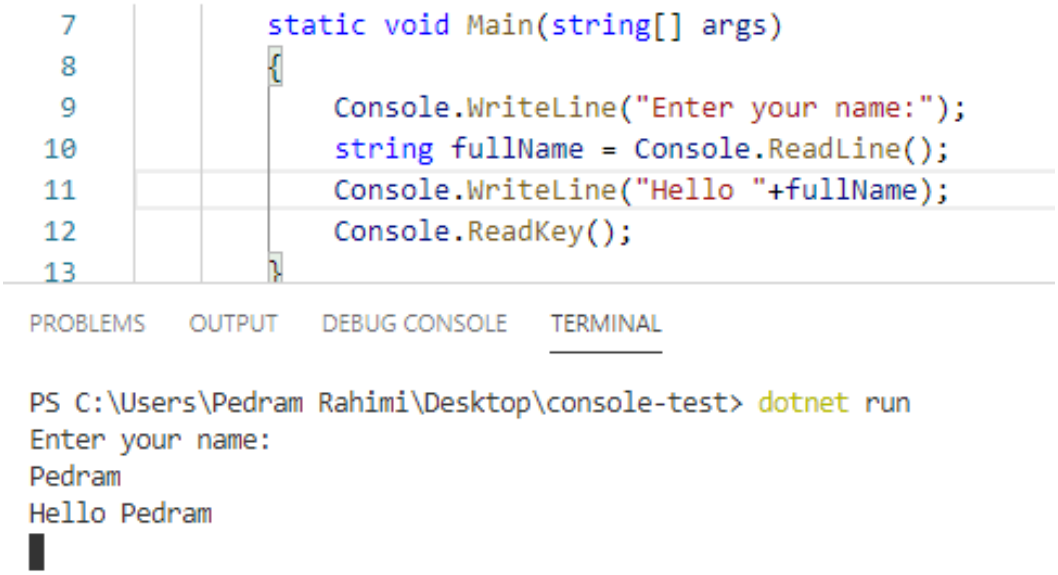
- **دستورات برنامه را برای اینکه اجرا شوند درون یک بلوک با نام Main می نویسیم.**
- **انتهای دستورات یک علامت ; که semicolon نام دارد، گذاشته می شود.**
- **دستورات حساس به حروف بزرگ و کوچک انگلیسی هستند.**

در برنامه ی قبلی انتهای دستور اول و بعد از سِمی کالُن نشانگر ماوس را قرار داده و اینتر بزنید تا به خط بعد بروید. به محض تایپ حروف Console لیستی از دستورات ظاهر می شوند که این همان خاصیت خودتکمیلی یا اینتلیسنس است. بعد از کلمه ی کنسول یک نقطه قرار دهید تا گزینه های دستور کنسول ظاهر شوند:



اکنون نوبت دستور جدیدی به نام **Console.ReadLine** است. از معنای انگلیسی آن روشن است که بر خلاف دستور قبلی به جای نوشتن مطلبی، آن را از کاربر گرفته و می خواند. اما چیزی که قرار است خوانده شود یک مقدار است که باید در جایی ذخیره شود. این جا که آن را در ریاضی به صورت نمونه هایی مانند x و y می شناختیم، در برنامه نویسی هم نامش متغیر است. متغیرها در سی شارپ انواع مختلفی دارند و باید حتماً نوع آنها تعریف شود. به طور پیشفرض هر چیزی که از طریق ReadLine خوانده شود به صورت متنی و همان String است حتا اگر عدد باشد. چون اعداد را هم به طور پیشفرض در حالت کاراکتری قبول می کند نه بر اساس مقدار عددی آنها.

حالا برنامه ی قبلی خود را کمی توسعه می دهیم و به جای نمایش پیام تنها، پیامی را همراه با یک مقدار متغیر مثل نامی که از ورودی گرفته می شود، نمایش می دهیم:



اول اینکه تم یا رنگ بندی ویرایشگر کد خود را می توانیم از طریق منوی Manage در گوشه ی سمت چپ و پایین یا فشردن کلیدهای ترکیبی **CTRL+T** انتخاب و تغییر دهیم. همچنین از منوی بالا **Terminal** را انتخاب کنید که باعث ظاهر شدن خط فرمان در زیر کدها شده و نیاز ما را از رجوع به کامند پرامپت برای اجرا رفع می کند.

با هم کدهای نوشته شده را بررسی می کنیم:

```
static void Main(string[] args)
{
    Console.WriteLine("Enter your name:");
    string fullName = Console.ReadLine();
    Console.WriteLine("Hello "+fullName);
    Console.ReadKey();
}
```

در اینجا اولین دستور یک عبارت را برای درخواست ورود نام در خروجی چاپ می کند. همانطور که فهمیده اید در مُد برنامه نویسی کنسول امکان استفاده از حروف و کاراکترهای فارسی نیست. پس فعلاً برنامه های خود را انگلیسی یا حداقل با کاراکترهای انگلیسی می نویسیم.

خط بعد همانطور در مورد متغیرها توضیح دادیم، با دستور string در ابتدا آماده ی تعریف متغیری به نام fullName می شویم از نوع استرینگ یا رشته ای. نوع رشته ای هم که همان متن یا غیرعددی است. این متغیر برابر با مقداری قرار داده شده است که با ReadLine از ورودی گرفته می شود.

در خط سوم از دستورات هم بعد از چاپ یک Hello و یک فاصله، متغیر fullName بیرون از دبل کوتیشن قرار داده شده. این کار به سی شارپ می گوید که متغیر است و به جای چاپ خود عبارت، باید مقدارش نمایش داده شود.

همانطور که در تصویر دیدید می توان در ترمینال پایین کدها دستور dotnet run را اجرا کرد. به عنوان مقدار ورودی نام خود را وارد کنید تا بعد از زدن اینتر، عبارت سلام و نام شما نمایش داده شوند.

هدف از به کار بردن این نمونه کدهای ساده علاوه بر جلو رفتن تدریجی در برنامه نویسی و فهم درست کاربرد دستورات، آشنایی مرحله به مرحله با روش کار است. به مرور متوجه ی پیچیده تر شدن برنامه خواهید شد.

محاسبات جدی

ساخت یک برنامه ی جدی شروع شد. محاسبات آن قدر مهم هستند که حتا نام کامپیوتر و شخصیت آن با این شکل از برنامه ها یعنی برنامه های محاسباتی شناخته شده است. در ساده ترین شکل یک برنامه ی محاسباتی اگر برنامه نویس باشید کافیست در موبایل یا کامپیوتر خودتان همانطور که در صفحات قبل دیدید، چند متغیر را تعریف، مقادیری را به عنوان ورودی برای آنها دریافت کرده و در نهایت به کامپیوتر دستور دهید که نتیجه ی محاسبات را طبق یک فرمول خاص چاپ کند:

```
// Console.WriteLine("Enter your name:");
// string fullName = Console.ReadLine();
double number1 = 15.3;
int number2 = 8;
```

```
Console.WriteLine( number1 + number2 );
Console.ReadKey();
```

اول از همه متوجه شدید که به جای پاک کردن خطوط برنامه ی قبلی با قرار دادن // در ابتدای آنها یعنی دبل اسلش این خطوط کامنت یا غیرفعال شده اند. شاید دوباره بخواهیم از آنها استفاده کنیم و با تغییرشان کاربرد جدیدی برای آنها بیابیم. هر چند در اصل، کامنت گذاری کارش نوشتن توضیح قبل از خطوط یا مقابل آنها است ولی به کمپایلر می فهماند که این خطوط جزو برنامه نیستند.

دستور **double** در این برنامه آماده است که یک نام متغیر از نوع اعشاری را تعریف کند.

دستور **int** مخفف integer یا عدد صحیح است و روشن است که فقط اعداد بدون اعشار را می توان به عنوان مقدار برایش تعریف کرد.

بقیه ی عملیات هم قابل فهم است و شک نیست که مثل دفعات قبل باید داخل عملگری (Function) با نام Main از لایه ای (class) با نام Program که نوعی طبقه بندی کدها در شی گرایی است، نوشته شوند. فعلاً به تشریح مفهوم کلاس و عملگر نپرداخته ایم و برای شناخت صحیح و ماندگار آن فعلاً فقط خواص آنها را بررسی خواهیم کرد.

نکته: میانبرهای کاربردی در وی اس کد

CTRL+K C : متن سلکت شده را به کامنت غیر قابل اجرا بدل می کند.

CTRL+K U : متن کامنت شده را از این حالت در می آورد و به کد بدل میکند.

Alt+Shift+↓ : کلید مکان نمای پایین باعث کپی گرفتن از خط جاری در زیرش می شود.

Alt+Shift+F : کدهای تایپ شده را به آرایش علمی و درست می چیند.

برنامه را اجرا کرده و نتیجه ی جمع دو عدد را در خروجی مشاهده کنید. با این روش مجموعه ای از عملیات مختلف دیگر را هم که با چهار عمل اصلی ریاضی یعنی جمع +، منها - ، ضرب * و تقسیم / قابل انجام است، می توانید بنویسید.

یکی از موارد جالبی هم که می توانید امتحان کنید، جمع کردن دو مقدار string است. خواهید دید که نتیجه ی جمع دو متغیر رشته ای در خروجی، کنار هم قرار گرفتن حروف آنهاست.

اما همانطور که شاید متوجه شده باشید این برنامه نمی تواند چندان کاربردی باشد. ما برنامه را فقط برای کسانی که سی شارپ بلد هستند نمی نویسیم. یک برنامه ی محاسباتی باید فقط مقادیر را از کاربر بگیرد و خودش محاسبات را انجام داده و خروجی را به نمایش بگذارد. نه آنکه مقادیر جدید را مجبور باشیم هر بار در کد برنامه وارد کنیم! با توجه به چیزی که در صفحات قبل دیده ایم می توان دریافت اعداد را هم مانند حروف آزمایش کرد:

```
static void Main(string[] args)
{
    Console.WriteLine("Enter number1:");
    double number1 = Console.ReadLine();

    Console.ReadKey();
}
```

سعی کنید ابتدا خطی را برای نمایش پیام ورود متغیر number1 بنویسید. تا اینجا مشکلی نیست. اما به محض آنکه خطی برای دریافت ورودی و انتساب آن به متغیری از نوع اعشاری بنویسید، حتا قبل از اجرا زیر دستور ReadLine خط قرمزی توسط ویرایشگر رسم می شود. اگر ماوس را روی این خط قرمز ببرید پیامی با این مضمون نمایش می دهد که شما قصد دارید مقداری رشته ای را به یک متغیر عددی نسبت دهید! اگر خاطرتان باشد قبل از این هم گفته ایم که مقادیر وارد شده در ReadLine همیشه به صورت string دریافت می شوند.

راه حل رفع این خطا تبدیل این مقدار به عدد است. float.Parse( ) دستوری است که مقادیر درون پرانتز خودش را به نوع اعشاری یا float تجزیه یا parse می کند. بنابراین کافیست که کل دستور بعد از علامت = را درون این دستور جدید قرار دهیم:

```
double number1 = float.Parse( Console.ReadLine() );
```

حالا ادامه ی برنامه را با همین راه حل می نویسیم:

```
Console.WriteLine("Enter number1:");
double number1 = float.Parse( Console.ReadLine() );
Console.WriteLine("Enter number2:");
double number2 = float.Parse( Console.ReadLine() );
Console.WriteLine("Sumation is: " + (number1 + number2) );
Console.ReadKey();
```

فقط در خطی که کد نمایش نتیجه در آن نوشته شده توجه کنید که عبارت متنی به اضافه ی حاصل جمع دو عدد ورودی شده اما این دو مقدار داخل پرانتز قرار داده شده اند. سعی کنید بدون پرانتز نتیجه ی برنامه را تست کنید. در این صورت خواهید دید به جای جمع دو عدد، حاصل جمع رشته ای یا کاراکترهای آنها نمایش داده می شوند! به این شکل خطاها bug یا خطاهای منطقی اما غیرقابل انتظار می گویند که اگر چه موجب توقف برنامه نمی شوند اما مطلوب نیستند.

در واقع کمپایلر سی شارپ به این دلیل که شما قبل از حاصل جمع دو عدد یک مقدار رشته ای "Sumation is:" را با آنها جمع کرده اید، کل فرآیند جمع را رشته ای تلقی می کند که قبل از این گفتیم جمع رشته ها، یعنی کنار هم قرار گرفتن آنها. اما وقتی داخل پرانتز قرار داده شوند ابتدا جمع دو متغیر عددی پردازش می شود که بالطبع درست انجام شده و نتیجه ی آن به صورت رشته ای به همراه عبارت متنی نمایش داده می شود.

حتا می توانستید بجای float.parse از دستور دیگری استفاده کنید:

```
Convert.ToDouble ( Console.ReadLine() );
```

دستور **Convert.ToDouble** انواع دیگری مثل **ConvertToString** دارد که همه ی مقادیر را به حروف بر می گرداند. اما **ConvertToInt** شکل های ۱۶، ۳۲ و ۶۴ بیتی دارد که تعداد ارقام عدد صحیح را در مقدار ذخیره شده تعیین می کند. فرق این مقادیر صحیح و تعداد ارقام این سه نوع عدد به شرح زیر است:

Int16 --> (-32,768 تا +32,767)

Int32 --> (-2,147,483,648 تا +2,147,483,647)

Int64 --> (-9,223,372,036,854,775,808 تا +9,223,372,036,854,775,807)

و به این ترتیب شما از این به بعد می توانید هر نوع برنامه ی محاسباتی را با هر فرمول یا مقدار ورودی بنویسید.

کنترل ساده ی خطا

برنامه ی قبلی را در نظر بگیرید. اگر چه این یک برنامه ی ساده و آزمایشی بود. اما فرض کنید یک برنامه ی جدی محاسباتی نوشته اید و قرار است نسخه ی اجرایی آن را برای کسی بفرستید. آن فرد که قرار است فقط با وارد کردن مقادیر اولیه، نتیجه ی محاسبات برنامه را ببیند و از سی شارپ هم هیچ اطلاعی ندارد به طور کاملاً غیرعمدی مقداری حرفی را به جای یکی از اعداد وارد می کند. اگر خودتان اکنون این کار را انجام دهید خواهید دید که برنامه بلافاصله متوقف شده و پیام خطایی نمایش داده می شود. این موضوع در پروژه های جدی به اعتبار برنامه لطمه می زند. برای همین برنامه نویسان در بسیاری از موارد اقدام به کنترل این خطاها می کنند:

Try { محل قرارگیری کدی که ممکن است خطا بدهد **}**

Catch { محل قرارگیری کدی که بعد از بروز خطا باید اجرا شود **}**

Finally { محل قرارگیری کدی که در هر صورت چه خطایی باشد و چه نباشد اجرا می شود **}**

حالا برنامه ی قبلی را که عملکرد آن را می دانیم، با کمک دستورات بالا می نویسیم:

```
using System;
namespace console_test
{
```

```
    class Program
    {
        static void Main(string[] args)
        {
            double number1 = 0;
            double number2 = 0;
            try
            {
                Console.WriteLine("Enter number1:");
                number1 = float.Parse(Console.ReadLine());
                Console.WriteLine("Enter number2:");
                number2 = float.Parse(Console.ReadLine());
            }
            catch
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("Enter just number ...");
                Console.ResetColor();
            }
            Console.WriteLine("Sumation is: " + (number1 + number2));
            Console.ReadKey();
        }
    }
}
```

در برنامه ی بالا متوجه شده اید که اول از همه تعریف اولیه ی متغیرها را بیرون از try انجام داده ایم. بعد از اجرای برنامه به محض اینکه یکی از مقادیر را به صورت حرفی وارد کنیم، بخش catch فراخوانی می شود. در آنجا جهت نمایش پیام مناسب ابتدا رنگ نوشتار را در اولین خط برابر قرمز قرار داده ایم. در خط بعد پیام مناسب برای ورود صحیح اعداد را نوشته ایم و در نهایت رنگ خطوط را Reset کرده ایم که به حالت عادی برگردد. همانطور که متوجه شدید نوشتن بخش finally هم الزامی نیست.

توجه داشته باشید که تعداد catch را هم به تعداد انواع خطاها می توان نوشت. کافیست مقابل آن داخل پرانتز نوع هر خطا را مشخص کنیم:

```
catch (FormatException) { … }
catch (DivideByZeroException) { … }
```

و البته بعد از نوع خطا می توان یک متغیر مثل ex هم قرار داد که کل پیام خطا در آن ذخیره شود و در صورت نیاز درون catch بتوان متن آن را داشت:

```
catch (FormatException ex) { … }
```

و البته می توان catch آخری را بدون پرانتز و تعریف نوع خطا نوشت که اگر شامل هیچ خطای تعریف شده نبود، آن قسمت اجرا شود.

شرط و تکرار

ممکن است ما نخواهیم که برنامه ی قبلی بعد از اولین محاسبه متوقف شود. معنای تکرار را به طور معمول با حلقه ها در یک زبان برنامه نویسی پیاده سازی می کنند. اما کنترل یک حلقه برای آنکه تا بی نهایت تکرار نشود با دستورات شرطی است. اگر قبل از سی شارپ زبانهای دیگری مثل Javascript و یا PHP را تجربه کرده باشید، خواهید دید که دستورات شرطی و حلقه ها در تمام زبانهای برنامه نویسی تقریباً مشابه بوده و یک جور کار می کنند. مثلاً برای یک حلقه ی ساده دستور زیر را پیاده سازی می کنیم:

```
while ( یک متغیر منطقی یا شرط )
        {عملیاتی که میخواهیم تکرار شود }
```

دانستن زبان انگلیسی بدون شک باعث پیشرفت ما در برنامه نویسی خواهد شد. همانطور که روشن است while به معنای "تا زمانی که" قرار است با زبانی نزدیک به زبان انسان، برنامه را پیش ببرد. درون پرانتز یک شرط یا متغیر منطقی از نوع bool را قرا می دهیم که همان Boolean است و می تواند فقط یکی از دو مقدار true یا false به خود بگیرد. این یعنی اگر فقط متغیر منطقی را درون پرانتز قرار دهیم، تا زمانی که مقدار آن true باشد حلقه تکرار می شود و می توان با پیشرفت برنامه درون حلقه در جایی این مقدار را false قرار داد تا برنامه از حلقه خارج شود.

حالا به سراغ شرط می رویم. دستور if به معنای "اگر" و else به معنای "در غیر اینصورت" یک روال شرطی کلاسیک را برای ما ایجاد می کنند:

```
if ( شرط مقایسه ای )
                { عملیاتی که در صورت درست بودن شرط انجام می شود }
  else { عملیاتی که در صورت درست نبودن شرط انجام می شود }
```

شرط و حلقه هر دو مباحث گسترده ای برای یادگیری هستند. اما در اینجا مجبور شدیم برنامه ی قبلی خود را فعلاً با یادگیری فرم های ساده ای قابل تکرار کنیم:

```
double number1 = 0;       // مقدار اولیه ی عدد اولی
double number2 = 0;       // مقدار اولیه ی عدد دوم
bool myCommand = true;    // مقدار اولیه ی دستور تکرار
string message = "";      // مقدار اولیه ی پیام کاربر
while (myCommand)         // شروع حلقه ی تکرار
{
    try
    {
        Console.WriteLine("Enter number1:");
        number1 = float.Parse(Console.ReadLine());
        Console.WriteLine("Enter number2:");
        number2 = float.Parse(Console.ReadLine());
```

```
    }
    catch
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Enter just number ...");
        Console.ResetColor();
    }
    Console.WriteLine("Sumation is: " + (number1 + number2));
    Console.WriteLine("Do you continue?(y/n)"); // پیام ادامه ی برنامه
    message = Console.ReadLine();                // دریافت پیام کاربر
    if (message == "y")    // مقایسه ی پیام کاربر با حالت تایید
    {
        myCommand = true;    // حالت تایید و تکرار حلقه
    }
    else
    {
        myCommand = false;   //حالت غیر تایید و خروج
    }
}
```

به محض اجرا و اولین محاسبه، این بار به جای خروج، از کاربر سوال می شود که آیا مایل به ادامه هستی؟ و اگر حرف y را وارد کند، حلقه ی ایجاد شده، برنامه را تکرار می کند، در غیر این صورت هر مقدار دیگر که وارد کند، برنامه متوقف خواهد شد.

فقط در بخش مقایسه ی پیام کاربر با حالت تأیید، داخل برنامه ها، به جای علامت = از علامت == استفاده شده است. در زبان سی شارپ و بیشتر زبانهای دیگر، یک مساوی فقط برای انتساب مقدار به کار می رود و در زمان شرط باید به این صورت باشد. اگر در شرط یک مساوی به کار رود کمپایلر سی شارپ فقط حالت bool یا منطقی را در نظر می گیرد و حالت های متنی و عددی را نمی تواند مقایسه کند. در واقع در == هم نوع متغیر و هم مقدار چک می شوند.

نکته: عملگرهای مقایسه ای

عملگرهای مقایسه در دستورات منطقی (Logical statement) به منظور تشخیص برابری یا تفاوت بین متغیرها (یا مقادیر) بکار می روند. با در نظر گرفتن این مثال که x = 5 و نوع آن عددی باشد، جدول زیر عملگرها را شرح می دهد. از کوچکتر و بزرگتر خالی صرف نظر شده:

عملگر	شرح	مثال	مقدار بازگشتی
==	برابر بودن مقدار	x == 5	true
		x == 8	false
!=	نابرابر بودن مقدار	x != 5	false
		x != 8	true
<=	کوچکتر و مساوی	x <= 8	true
>=	بزرگتر و مساوی	x >= 8	false

در زمان مقایسه ی مقادیر رشته ای یکی از پُر کاربردترین دستورات، تبدیل رشته های انگلیسی به حروف کوچک یا بزرگ است:

```
message = message.ToLower();
```

خاصیت دستور بالا این است که اگر کاربر به هر دلیلی دستور را با حروف بزرگ هم وارد کرده باشد بتوان در هنگام مقایسه آن را به حروف کوچک بدل کرد و مقایسه را انجام داد. همینطور ToUpper. هم برای تبدیل به حروف بزرگ انگلیسی به کار می رود.

اگر تبدیل بالا را در برنامه انجام نداده بودید باید شرط خود را به صورت زیر می نوشتید:

```
if (message == "y" && message =="Y") { .. }
```

این یعنی هم y و هم Y را چک کن و در صورتی که هر دو شرط برقرار بودند عمل کن. این کار با ترکیب منطقی and که به صورت && نوشته شده انجام می شود. مقایسه ی or هم با || انجام می شود که در کیبورد استاندارد کاراکتر بقل یا بالای کلید اینتر با Shift است. در حالت or اگر یکی از شروط هم برقرار باشد، شرط قبول است یعنی این "یا" آن.

البته همانطور که فهمیده اید، این عملگرها برای مقایسه ی تنها دو شرط مناسب هستند. فرض کنید ما در برنامه ورودی های زیادی داشته باشیم که لازم باشد هر کدام را چک کنیم. مثلاً کاربر عددی را وارد کرده باشد و ما بخواهیم آن را به یکی از روزهای هفته نسبت داده و آن روز را نمایش دهیم (یک برنامه ی جدید):

```
string message = "";
Console.WriteLine("Enter number of day: ");
message = Console.ReadLine();
if (message == "1")
{
    Console.WriteLine("Shanbeh!");
}
if (message == "2")
{
    Console.WriteLine("YekShanbeh!");
}
Console.ReadKey();
```

البته همانطور که می بینید فقط برای دو روز اول هفته کدها را نوشته ایم. یکی از راههای بهینه تر استفاده از شرط، بهره گیری از دستور جدید به شکل زیر است:

```
int message = 0;
Console.WriteLine("Enter number of day: ");
message = Convert.ToInt32(Console.ReadLine());
switch (message)
{
    case 1:
        {
            Console.WriteLine("Shanbeh");
            break;
        }
```

```
            case 2:
                {
                    Console.WriteLine("YekShanbeh");
                    break;
                }
            default:
                {
                    Console.WriteLine("HiChi!");
                    break;
                }
        }

        Console.ReadKey();
```

این شکل جدید از چک کردن شرط به صورت switch همانطور که به نظر می رسد از سرعت بیشتری نسبت به if برخوردار خواهد بود. توجه داشته باشید که بخش **break** باید بعد از چک کردن هر case مورد استفاده قرار گیرد تا در صورت برقرار بودن شرط، ادامه ی چک کردن در داخل سوئیچ متوقف شود. البته شما می توانید آن را به صورت حرفی یا رشته ای هم بنویسید و لازم نیست مثل این برنامه ی نمونه حتماً کیس های شما عدد باشند. یعنی می توانند نامها یا رشته ها باشند که البته داخل دبل کوتیشن می روند. شما کیس های ۳ و ۴ و .. را برای دیگر روزهای هفته در این برنامه بنویسید. در نهایت اگر هیچ موردی هم با هیچ کیس منطبق نباشد، بخش **default** فراخوانی می شود.

آرایه و حلقه

در حلقه ی while متوجه شدید که تنها یک شرط در پرانتز مشخص می کند که این حلقه چند بار تکرار شود. بعد با کنترل مقدار آن در هر جا درون حلقه می توانستیم تکرار حلقه را متوقف کنیم. گاهی لازم است که به طور دقیق از همان ابتدا تعداد تکرار را در حلقه مشخص کنیم. به این ترتیب به شکل پیشرفته تر و گویاتری از دستورات برای ایجاد این حلقه ها نیاز داریم:

```
for (مقدار اولیه; شرط مقدار; افزایش مقدار ) {عملیاتی که میخواهیم تکرار شود }
```

ممکن است توضیح بالا کمی گنگ باشد. بنابراین آن را با یک مثال آزمایش می کنیم:

```
for (int i = 1; i <= 5; i=i+1)
{
    Console.WriteLine(i);
}
Console.ReadKey();
```

برنامه ی کوتاه بالا، یک حلقه ی ساده ایجاد می کند که مقدار اولیه ی متغیر i در آن یک است. بعد چک می کند که i کوچکتر و مساوی ۵ باشد و مرحله ی سوم i یکی یکی اضافه می شود. در وسط حلقه این مقدار چاپ می شود. به این ترتیب با اجرای برنامه ی فوق اعداد یک تا پنج چاپ می شوند. این برنامه را با حلقه ی While هم می توانستیم بنویسیم ولی نه به این یکپارچگی:

```
int i = 1;
while(i<=5){
    Console.WriteLine(i);
    i++;
}
```

خروجی و منطق هر دو برنامه یک جور است اما روش نوشتن کدها متفاوت. در واقع کد اول با for تمیزتر و قابل فهم تر است. در ضمن ++ بعد از متغیر هر بار عدد یک را به آخرین مقدار i اضافه می کند. همینطور -- یعنی دو علامت منها (ماینس ماینس) هم اگر به کار روند هر بار یک مقدار کم می کنند. در ضمن در حلقه ی for بخش سوم یعنی افزایش مقدار اجباری نیست. یعنی می توانید داخل حلقه آن را تعریف کنید. فقط مراقب باشید که حلقه های بی انتها نسازید.

نکته: نامهای انگلیسی دربرگیرنده ها

( )	**پرانتز**	Round brackets	یا Parentheses
{ }	**بریس**	Curly brackets	یاBraces
[ ]	**براکت**	Square brackets	یا Simply brackets

سر و کله ی آرایه ها زمانی پیدا می شوند که بخواهیم این مقادیر را درون متغیرها نگهداری کنیم و از دست ندهیم. مثلاً در حلقه هایی که دیدیم مقادیر قبلی i همه از بین می رفتند. در اینجا مثال جدید خود را با مقدار دهی به یک آرایه شروع می کنیم:

```
int[] numbers = {12, 5, 35, 80, 2, 56};
for (int i = 0; i <= 5; i=i+1)
{
    Console.WriteLine(numbers[i]);
}
```

و همانطور که بعد از اجرا خواهید دید مقادیر ذخیره شده در آرایه ی numbers که تنها فرقش با متغیر در زمان ساخت، استفاده از [ ] بوده و در زمان مقدار دهی هم مقادیر با { } و جدا سازی توسط ویرگول وارد شده اند، به ترتیب نمایش داده می شوند.

در واقع اندیس های آرایه که از صفر شروع می شوند (اولین عضو شماره ی صفرم است) با number[0] و آخرین عضو با numbers[4] فراخوانی می شوند که جمعاً ۵ عدد هستند. اینجا [ ] مقابل int قرار داده شده است. می توانید آن را مقابل string هم قرار داد و فقط مقادیر حرفی یا رشته ای داخل جفت کوتیشن یا دبل کوتیشن و درون { } قرار می گیرند.

در مثال قبلی مشخص بود که ۶ عدد مقدار داریم و به محض تعریف آرایه این شش مقدار را هم به آن دادیم. اما در برنامه های کاربردی تر معمولاً این تعداد مشخص نیست. بنابراین شکل تعریف آرایه مشخص نیست. مثلاً ممکن است که بخواهیم لیستی از افراد را دریافت کنیم. در این صورت اول تعداد افراد را از ورودی می گیریم به نام personsNumber و بر آن اساس شیء آرایه را با دستور new بازسازی کرده و به متغیری مانند persons نسبت می دهیم.

```
Console.WriteLine("Enter number of Persons: ");
int personsNumber = int.Parse(Console.ReadLine());
string[] persons = new string[personsNumber];
```

```
for (int i = 0; i < personsNumber; i++)
{
    Console.WriteLine("Enter Person number " + (i + 1) + " name: ");
    persons[i] = Console.ReadLine();
}
Console.WriteLine("The persons are:");
foreach (string name in persons)
{
    Console.Write(name + ", ");
}
Console.ReadKey();
```

بعد از دریافت تعداد افراد حلقه ای به تعداد افراد ساخته ایم که نامها را به ترتیب از ورودی بگیرد. در اینجا با یک حلقه ی جدید هم ویژه ی نمایش مقادیر درون آرایه آشنا می شویم. foreach فرقش با for در این است که دو بخش مقدار اولیه و افزایش مقدار را ندارد. بلکه یک متغیر اولیه بسته به نوع آرایه که عددی یا رشته ای باشد گرفته و خودش به طور خودکار به تعداد اعضای آرایه لوپ میزند تا تمام مقادیر را خوانده و به ترتیب در این متغیر اولیه که اینجا با نام name تعریف کرده ایم قرار دهد:

```
foreach (متغیر معمولی از جنس آرایه in آرایه ی مورد نظر ) { عملیاتی که میخواهیم }
```

نکته ی دیگر فرق Write با WriteLine است که بدون رفتن به خط جدید، مقادیر را نمایش می دهد. در این مثال از try .. catch پرهیز شده که مفاهیم جدید فقط منتقل شوند. با اجرای برنامه ابتدا تعداد افرادی را که قصد ورود نام آنها را داریم پرسیده می شود و بعد به همان تعداد نام دریافت می شود و سرانجام نامها کنار هم نمایش داده می شوند. این تنها یک نمونه از کار و نحوه ی دسترسی به لیست ها (آرایه ها) را به ما یاد می دهد. در برنامه های جدی تر ممکن است عملیات پیچیده تری نسبت به خواندن و نمایش تنها روی این داده ها داشته باشیم.

کلاس جدا و عملگر

کمی در مورد ساختار کلی یک فایل سی شارپی با پسوند .cs بدانیم:

```
using System;
namespace my-project
{
    class Program
    {
        static void Main(string[] args)
        {
            ...
        }
    }
}
```

از خط اول، using به کتابخانه ای اشاره دارد که دستورات درون این صفحه از آن بهره می گیرند. با توجه به اینکه ما فعلاً از دستورات هسته ی اصلی سی شارپ بهره برده ایم، فقط یک کتابخانه به نام system را صدا زده ایم. ممکن است در صفحات دیگر به تعداد کتابخانه های لازم عمل using به همراه نام آن کتابخانه ها را داشته باشیم.

خط بعد فضای نام پروژه یا محدوده ی کار اصلی که به عنوان بلوک اصلی هم شناخته می شود و نام پروژه در مقابل آن نوشته شده را داریم. اگر پوشه یا پروژه های دیگری را داشته باشیم، فضای نام جدیدی بالطبع خواهیم داشت.

پس از اینها حالا کلاس Program یا کلاس اصلی برنامه است. درون بلوک فضای نام می توانیم بلوک یا کلاس های دیگری هم تعریف کنیم و باید بدانیم که متغیرها و عملگرهای آن به صورت پیشفرض مختص همان کلاس محسوب شده و در کلاس های دیگر تعریف شده به حساب نمی آیند. مثلاً عملگر خودکار ( )Main که با پرانتزهای بعد از آن نام عملگر به خود می گیرد در واقع مخصوص بلوک Program است.

همانطور که فهمیده اید، بسیاری از دستورات هم که در مقابلشان ( ) قرار دارد یک عملگر (تابع) محسوب می شوند که در سی شارپ با توجه به قرارگیری درون کلاس ها به آنها متد می گویند.

یعنی عملگر درون کلاس نامش method است. در واقع این دستورات خودشان مجموعه ای از چند دستور بوده اند که در کتابخانه ی کد دات نت به صورت یک دستور مستقل شناخته شده اند.

عملگرها به طور ویژه کارشان جلوگیری از تکرار دستورات تکراری است. به این معنی که مجموعه ی چند دستور را درون یک عملگر قرار داده و برای کاربردهای تکراری فقط آن عملگر را که شامل چندین دستور است صدا می زنیم و به این ترتیب از نوشتن مجدد آن چند دستور خودداری می کنیم.

ممکن است این تعاریف به این شکل گنگ باشند و مجبور باشیم با مثال آنها را بشناسیم. البته عملگرها همیشه پارامتر نمی گیرند و مانند عملگر Main ممکن است خروجی مشخصی هم نداشته باشند که در این صورت قبل از آنها عبارت void نوشته شده وگرنه باید با دستور return حتماً یک مقداری را برگردانند.

ساده ترین مثالی که می توان از یک عملگر دست ساز در سی شارپ زد را ببینید:

```
static void SayHello(string name)
{
    Console.WriteLine("Hello " + name);
}

Console.WriteLine("Enter First name: ");
string name1 = Console.ReadLine();
SayHello(name1);

Console.WriteLine("Enter secound name: ");
string name2 = Console.ReadLine();
SayHello(name2);

Console.ReadKey();
```

همه ی خطوط بالا درون عملگر Main( ) که می دانیم به صورت خودکار اجرا می شود، باید نوشته شوند. اما همانطور که می بینید به همان روش تعریف عملگر Main یک عملگر به نام SayHello ساخته ایم که یک آرگومان یا پارامتر ورودی از جنس string درون پرانتز مقابلش

گرفته و عملیات چاپ سلام را با آن نام انجام می دهد. بعد از آن هم دو بار در کدهای مختلف و دو جای مختلف برنامه نامی را از ورودی گرفته ایم و درون متغیرهای متفاوتی ریخته ایم و بعد از آن با فراخوانی تابع سلام کردن و دادن آن متغیر به آن عملیات سلام کردن را انجام داده ایم بدون اینکه دستورات درون تابع را دوباره نوشته باشیم. هر چند این مثال چندان حرفه ای نیست اما به خوبی کارکرد عملگر را نشان می دهد. شما می توانید عملگرهایی بنویسید که داخل پرانتز آنها چیزی نباشد و در واقع آرگومان ورودی نداشته باشند و فقط درون آنها چند دستور ثابت نوشته شود مثل ReadLine( ). که به عنوان یک متد داخلی سی شارپ فقط یک رشته از ورودی می گیرد. یا اینکه عملگرهایی بنویسید که یک یا چند ورودی داشته باشند و با توجه به این مقادیر گرفته شده عملیاتی را انجام دهند.

اما اگر به خاطر داشته باشید یک عملگر مانند ToLower( ). تمام حروف یک رشته را به حروف کوچک بدل می کرد. در واقع این دستور خودش یک عملگر نوشته شده در کتابخانه های دات نت است که ما را از نوشتن دستوراتی پیچیده و تکراری برای تبدیل حروف یک متغیر رشته ای به حروف کوچک بی نیاز کرده است و یک عملگر داخلی سی شارپ محسوب می شود.

حالا می فهمیم که عملگر Main( ) هم با توجه به آرایه ی رشته ای که داخل پرانتز مقابلش به عنوان آرگومان ورودی تعیین شده، مجموعه ای از دستورات را به عنوان تعدادی مقادیر رشته ای دریافت کرده و طبق عملیاتی که در هسته ی سی شارپ برای این عملگر یعنی Main نوشته شده، تک تک این دستورات را به اجرا می گذارد!

برای اینکه مفهوم یک کلاس و دسترسی های آن را هم بدانیم، عملگر SayHello را از Main خارج می کنیم و مانند نمونه ی زیر دوباره برنامه را اجرا می کنیم:

```
class Program
{
    static void SayHello(string name)
    {
        Console.WriteLine("Hello " + name);
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Enter First name: ");
```

```
            string name1 = Console.ReadLine();
            SayHello(name1);

            Console.WriteLine("Enter secound name: ");
            string name2 = Console.ReadLine();
            SayHello(name2);

            Console.ReadKey();
        }
}
```

همانطور که اینجا می بینید، عملگرها و متغیرهای درون یک کلاس برای هم قابل شناسایی هستند. اما اگر خارج از کلاس Program متغیر یا عملگری تعریف کنیم، دیگر به صورت پیشفرض درون این کلاس به آنها دسترسی نداریم. این موضوع یکی از پایه ای ترین مفاهیم شیءگرایی است.

نکته: آیین قراردادی نامگذاری ها برای کدنویسی تمیز

نام کلاس و عملگرها به صورت **PascalCase** : یعنی به هم چسبیده، حروف اول بزرگ.

نام فایل و فولدر به صورت **lower-case** : حروف همه کوچک، بین کلمات خط تیره یا دَش.

نام متغیر و آرایه به صورت **camelCase** : یعنی همه ی حروف به هم چسبیده، حرف اول اولین کلمه کوچک و حرف اول بقیه ی کلمات به صورت بزرگ نوشته می شوند.

در واقع class به فارسی یعنی رده بندی و یک مفهوم انتزاعی (خیالی) است. در یک برنامه ممکن است بخش های مختلفی داشته باشیم که هر بخش یک رده بندی یا کلاس خاص خودش باشد. به این ترتیب ممکن است مثلاً رده بندی اشخاص با عنوان کلاس Person شامل بخش های ویژه ی خود باشد که در کلاس های دیگر در دسترس نباشند. این بخش ها همان دو بخش عملگر یا متغیرها هستن. یعنی در مثال Person که یک کلاس مثلاً در سیستم مدیریت پرسنل محسوب می شود، ممکن است عملگرهایی مانند عملگر محاسبه ی کارکرد، عملگر ذخیره ی غیبت ها و عملگرهایی از این دست نوشته شوند. همینطور متغیرهایی که به عنوان صفات این کلاس

تعریف می شوند از جمله نام، شماره ی پرسنلی و نظایر اینها هستند. همه ی این دسته بندی های برای خواناتر شدن کدنویسی و امکان توسعه ی سریع هستند.

حالا لازم است حالت مقدار برگردان عملگر را هم بشناسیم. یعنی دیگر عبارت void در مقابل آن نوشته نشود. در این صورت لازم است دستور return در بدنه ی آن نوشته و یک خروجی داشته باشد. به عنوان مثال یک عملگر به نام Summation می نویسیم که دو عدد از ورودی گرفته و جمع آنها را برگرداند:

```
static int Summation(int x, int y)
{
    return x + y;
}
static void Main(string[] args)
{
    Console.WriteLine(Summation(50, 45));
    Console.WriteLine(Summation(20, 4));
    Console.WriteLine(Summation(500, 12));
    Console.ReadKey();
}
```

همانطور که می بینید نوع این عملگر را هم int قرار داده ایم. در این مثال فقط مقادیر ثابت را طی سه فقره فراخوانی عملگر محاسبه و نمایش داده ایم. البته می توانیم به جای مقادیر ثابت متغیرهایی از جنس int هم قرار دهیم. از طرفی وقتی می گوییم عملگر مقداری را بر می گرداند به این معنی است که هم می توان آن را با دستوری شبیه WriteLine نمایش داد و هم این مقدار را به متغیر هم نسبت داد. حالا می فهمیم که چرا عملگر وقتی مقداری بر می گرداند و void نیست، نوع هم باید داشته باشد مثل int تا قابل انتساب باشد.

نکته ی باقیمانده در مورد عملگرها این است که مانند متغیرها در یک کلاس نمی توانند اسامی تکراری داشته باشند. اما اگر آرگومان های ورودی آنها متفاوت باشد می توانند اسامی تکراری پیدا کنند. مثلاً عملگری که دو مقدار int قبول کند می تواند نامی مشابه عملگری داشته باشد که یک مقدار int قبول می کند. مشخص است که هنگام استفاده از آنها نیز با فرم مقداردهی به آنها قابل تفکیک می شوند.

حالا برای اینکه با سطوح دسترسی در کلاسها هم آشنا شویم مثال زیر را بررسی می کنیم:

```
namespace my-project
{
    class MyClass
    {
        public static int Summation(int x, int y)
        {
            return x + y;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(MyClass.Summation(50, 45));
            Console.ReadKey();
        }
    }
}
```

در اینجا با استفاده از مثال های قبلی، عملگری را برای جمع کردن دو عدد داریم که درون بلوک جدیدی با نام کلاس MyClass تعریف شده است. با این تفاوت که جلوی آن هم عبارت public به معنای عمومی قرار داده شده تا در کلاس های دیگر قابل دستیابی باشد. همانطور که گفته شده بود تعریف پارامتر (متغیر) یا متد (عملگر) درون کلاسی باعث می شود که به طور پیشفرض درون کلاس های دیگر در دسترس نباشد یا به عبارت دیگر سطح دسترسی آن private باشد.

در ضمن همانطور که می بینید نحوه ی صدازدن عملگر درون یک کلاس از کلاس دیگر با قرار دادن نام آن کلاس در ابتدای آن میسر شده و در این مثال به صورت زیر نوشته شده:

```
MyClass.Summation(..)
```

در اینجا همانطور که می بینید هر دو کلاس Program و MyClass در یک فایل به نام Program.cs نوشته شده اند. راه بهتر این است که شما فایل جدیدی با نام MyClass.cs ایجاد کنید و کد کلاس MyClass را در حالی که مانند کلاس Program ذیل و زیرمجموعه ی فضای نام پروژه ی شماست، به این فایل جدید منتقل کرده و از فایل Program.cs آن را پاک کنید:

```
using System;

namespace my-project
{
    class MyClass
    {
        public static int Summation(int x, int y)
        {
            return x + y;
        }
    }

}
```

بله کدهای بالا مربوط به فایل جدید **MyClass.cs** هستند که کنار Program.cs قرار دارد و در اجرای پروژه هیچ تفاوتی ایجاد نمی کند بلکه تنها برای یافته شدن سریعتر و پرهیز از قاطی شدن بی مورد کلاس ها آنها را در فایل جدید ایجاد می کنیم.

همانطور که می بینید نام کلاس جدید باید با فایل آن یکسان بوده و زیرمجموعه ی همان فضای نام پروژه که در فایل Program.cs داشته ایم، باشد.

حالا تنها کلمه ی **static** را به معنای نمونه ی ثابت باید بشناسیم. اگر این کلمه را از جلوی عملگر Summation حذف کنیم، عملگر ما تبدیل به یک عملگر صد درصد انتزاعی و تعریفی خواهد شد که برای استفاده از آن باید حتماً از روی آن نمونه سازی شود.

برای توضیح بهتر می توان گفت که مثلاً کلاسی با نام اتومبیل می تواند یک کلاس غیر static باشد که به تنهایی مفهومی برای استفاده نداشته باشد و اگر بخواهیم به عملگرهای آن مقدار داده و با آنها کار کنیم باید یک نمونه اتومبیل مثل پژو از روی آن اول بسازیم و بعد از آن به جای قرار دادن نام اتومبیل در ابتدای عملگری مثل حرکت یا تعویض دنده یا نظایر آن، نام پژو را که یک شیء ساخته شده از روی کلاس اتومبیل است قرار دهیم.

با توجه به مثال قبل بعد از حذف static از مقابل عملگر Summation درون کلاس MyClass حالا برای استفاده از آن در کلاس Program به شکل زیر باید از عملگر مربوطه استفاده کنیم:

```
class Program
{
    static void Main(string[] args)
    {
        MyClass MyClass1 = new MyClass();
        Console.WriteLine(MyClass1.Summation(50, 45));
        Console.ReadKey();
    }
}
```

همانطور که می بینید، ابتدا نام کلاس را می آوریم و بعد نام نمونه (instance) که می خواهیم از روی این کلاس بسازیم یعنی MyClass1 را آورده و آن را برابر نمونه ی جدید از MyClass قرار می دهیم که با دستور new این کار صورت گرفته.

حالا دیگر به جای MyClass که عملگرش به صورت غیر استاتیک است و قابل نمونه سازی بوده، از نمونه ی ساخته شده با نام MyClass1 به همان صورت استفاده کرده ایم و عملگرها و متغیرهای درون آن کلاس را می توانیم اینجا درون کلاس Program با شی یا نمونه ی آن جدید آن کلاس که از روی کلاس ساخته ایم صدا بزنیم. یعنی به محض تایپ کردن نام MyClass1 و قرار دادن یک نقطه بعد از آن در ویرایشگر VScode می بینیم که عملگر Summation به صورت خودکار نمایش داده شده و به ما پیشنهاد داده می شود.

همچنین باید بدانید **اگر از کلاسی که عملگر یا متغیرهای درون آن استاتیک هستند نمونه سازی کنید، دیگر به آن عملگر یا پارامترها دسترسی ندارید چون موارد استاتیک تحت نام خود کلاس مفهوم پیدا کرده و در نمونه ها نمودی ندارد.**

برای فهم این موضوع تصور کنید در کلاس اتومبیل که قرار است از آن نمونه سازی خودروهای مشخص انجام شود عملگر پرداخت بیمه، استاتیک و برای همه ی نمونه ها ثابت باشد. در این صورت دیگر فارغ از اینکه نمونه ی ساخته شده چه باشد در یک نمونه ی خاص قابل دسترس و تغییر مقدار نیست و به جای آن عملگرهای غیر استاتیک مثل حرکت کردن یا ایستادن را می توان در دسترس داشت.

در واقع استاتیک کردن برای زمانی است که نخواهیم در نمونه های ساخته شده از کلاس موارد استاتیک شده مقدار جدید بگیرند و مقادیر آنها یک بار برای همیشه تعریف شوند. حتا می توانید عبارت static را ابتدای عبارت class قرار دهید. به این ترتیب تمام عملگرها و متغیرها را هم باید استاتیک معرفی کرده و بدانید از چنین کلاسی امکان نمونه سازی دیگر وجود ندارد. مثلاً در کلاس استاتیک مدرسه همه ی مقادیر از مشخصات مدرسه ی مورد نظر که برنامه برای آن نوشته شده، ثابت هستند. اما در کلاس غیر استاتیک دانش آموز به تعداد دانش آموزان می توانیم نمونه سازی کرده و مقادیر جدید پاس کنیم.

سازنده ی کلاس

گاهی لازم است هنگام نمونه سازی از روی کلاس این اجبار را قرار دهیم که برخی متغیرها یا همان پارامترهای یک کلاس حتماً مقدار دهی شوند. برای این کار باید ابتدا درون یک کلاس، عملگری با نام همان کلاس بسازیم که به آن سازنده ی کلاس یا Constructor گفته می شود. برای این منظور با توجه به مثال قبل ابتدا ساختار کلاس MyClass را تغییر می دهیم:

```
class MyClass
   {
       public int x;
       public int y;
       public MyClass(int X, int Y)
       {
           this.x = X;
           this.y = Y;
       }
```

```
        public int Summation(int x, int y)
        {
            return x + y;
        }
    }
```

حتماً باید نام سازنده ی کلاس که یک عملگر درون آن کلاس است و خود کلاس یکسان باشند. حالا درون آرگومان های عملگر سازنده ی کلاس نام دو متغیر را قرار داده ایم که به محض نمونه سازی از روی کلاس باید توسط کلاس مقصد وارد شوند. این دو متغیر را با نام های بزرگ X و Y مشخص کرده ایم. درون عملگر سازنده ی کلاس اما خواسته ایم که پارامترها ی کوچک x و y که در ابتدای کلاس معرفی شده اند و ربطی هم به پارامترهای درون عملگر Summation ندارند بعد از مقدار گرفتن توسط سازنده، برابر با مقدار دریافت شده توسط سازنده شوند. عبارت this. قبل از آنها اشاره به همان کلاس جاری و پارامترهای تعریف شده در آن دارند. حالا در کلاس پروگرام با کد قبلی دچار خطا شده ایم چون نمونه ای که از روی MyClass ساخته ایم، مقادیر ورودی ندارد. برای این منظور کافیست کد موجود در کلاس Program را به صورت زیر تغییر دهیم:

```
MyClass MyClass1 = new MyClass(0,0);
```

در اینجا همینطور فقط برای رفع خطا مقادیر صفر را وارد کرده ایم. اما این مثال به خوبی تأثیر کانستراکتور یا سازنده ی کلاس را نشان می دهد. با توجه به تعریفی که از استاتیک داشته ایم، اگر عبارت static در ابتدای سازنده قرار داده شود، فقط یک بار در نمونه های ساخته شده فراخوانی می شود. یعنی هر تعداد از کلاس مورد نظر نمونه ی جدید بسازیم، به هر حال یک بار بیشتر سازنده اش اجرا نمی شود.

ارث بری کلاس ها از هم

اینجا با تمام توان سعی شده که مفاهیم شیءگرایی به ساده ترین شکل ممکن بیان شوند. همانطور که زبانهای برنامه نویسی سطح بالا خودشان را به زبان انسان نزدیک تر کرده اند، شیوه ها و الگوهای کدنویسی هم مطابق زندگی انسانها تعریف شده اند. **ارث بری به زبان ساده یعنی**

کلاسها بتوانند از عملگرها و متغیرهای یکدیگر بدون واسطه استفاده کنند. اگر چه این تعریف بسیار کامل و گویا است اما می توان آن را در یک مثال هم آزمایش کرد:

```
class Father
{
    public static double Summation(double x, double y)
    {
        return x + y;
    }
    public static string message = "Hi!";
}

class Girl
{
    public static double Minus(double x, double y)
    {
        return x - y;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(Father.Summation(12, 5));
        Console.WriteLine(Father.message);
        Console.WriteLine(Girl.Minus(15, 10));
        Console.ReadKey();
    }
}
```

برنامه ی نمونه ی بالا را بنویسید. همانطور که می بینید در بلوک فضای نام، قبل از کلاس Program دو کلاس دیگر به نامهای Father و Girl تعریف شده اند. هیچ مورد جدیدی در این برنامه قرار داده نشده است. ضمن اینکه می بینیم در زمان اجرا به راحتی عملگرها و متغیرهای مربوط به هر کلاس با نام آن کلاس در ابتدای آنها فراخوانی و اجرا شده اند.

حالا تغییر کوچکی تنها در نام کلاس Girl می دهیم:

```
class Girl:Father
```

و این بار برای فراخوانی پارامتر و متدهای کلاس پدر در کلاس پروگرام، فقط از فرزند استفاده می کنیم:

```
Console.WriteLine(Girl.Summation(12, 5));
Console.WriteLine(Girl.message);
Console.WriteLine(Girl.Minus(15, 10));
```

توجه دارید که عملگر Summation و متغیر message هیچ کدام در کلاس فرزند نبوده اند. اما به دلیل تغییری که در ارث بری این کلاس از کلاس والد پیدا کرده، به این دو بخش دسترسی دارد. همانطور که دیدید شیوه ی انجام این کار بسیار در سی شارپ ساده بود. ممکن است در زبان های برنامه نویسی دیگر برای این کار دستور یا آیین نگارش خاصی وجود داشته باشد. هر چه که باشد مفهوم آن با چیزی که در اینجا دیدید یکسان است. و البته سی شارپ از نظر پیاده سازی مفاهیم شیء گرایی زبانی به نسبت کامل تر و مدرن تر از بقیه ی زبانهاست.

مایکروسافت به بیان کامل اصول شیءگرایی در سی شارپ بسنده نکرده و با ارائه ی زبان Typescript که پس از کمپایل به زبان Javascript نسخه ی ۵ بدل می شود، کمبودهای شیءگرایی در جاوااسکریپت۵ را هم حل کرده است. بنابراین یادگیری موفق سی شارپ می تواند در فراگیری تایپ اسکریپت نیز که در وب کاربرد ویژه ای دارد، به شما کمک کند.

تا این لحظه دو اصل اساسی شیء گرایی شامل:

- **کپسوله کردن** (Encapsulation) که همان تفکیک دسترسی عملگرها و متغیرها در کلاس های متفاوت و بی تأثیر بودن آنها روی یکدیگر بود و

- **ارث بری** (Inheritance) را به خوبی فهمیده اید.

در بحث کپسوله سازی اما تا این لحظه فقط با دو صفت public به معنای دسترسی عمومی و private به معنای دسترسی خصوصی آشنا شدید. **اگر در مقابل نام کلاس هیچ عبارتی قرار داده نشود، این کلاس به صورت پیشفرض پابلیک در نظر گرفته شده و در کلاس های دیگر قابل مشاهده است.**

اما اگر در مقابل آن private را قرار دهید به هیچ عنوان نه خودش و نه عملگرها یا پارامترهایش در هیچ کجای دیگر خارج از آن کلاس در دسترس نخواهند بود. توجه داشته باشید که **در یک فضای نام یا پروژه نمی توان خود کلاس را با private به طور کامل محدود کرد اما عناصر داخل آن مثل عملگرها و پارامترها را می توان.** چون مشخص است که وقتی کل یک کلاس محدود شود دیگر هیچ استفاده ای نمی توان از آن کرد! اما در مثال قبلی مثلاً اگر در کلاس Father پارامتر message به صورت زیر تغییر کند:

```
 private static string message = "Hi!";
```

خواهید دید که خط قرمزی زیر فراخوانی آن توسط کلاس فرزند دیده خواهد شد:

```
Console.WriteLine(Girl.message);
```

چون سطح دسترسی این عنصر از کلاس والد آن محدود به خود آن کلاس شده و خصوصی تعریف شده است.

حالا می توانیم با اصطلاح جدید دیگری در این رابطه آشنا شویم. اما نوع **protected** به معنای محافظت شده اگر ابتدای class یا عناصر مربوط به کلاس قرار گیرد، فقط و فقط برای کلاس هایی در دسترس است که از این کلاس ارث بری داشته باشند. برای توضیح دقیق تر به آخرین مثال در چند سطر بالاتر بر می گردیم و private را به protected تغییر می دهیم:

```
protected static string message = "Hi!";
```

اما باز خط قرمز زیر message در هنگام فراخوانی برای چاپ از بین نرفته است. چرا؟ بله این تنظیم باعث شده که message از کلاس Father فقط و فقط درون کلاس **Girl** در دسترس باشد. یعنی عملگرهای داخل کلاس فرزند می توانند از آن بهره بگیرند اما در کلاس Program که عمل فراخوانی برای چاپ را انجام داده ایم در دسترس نیست. برای رفع این خطا می توانیم پارامتر جدیدی در کلاس فرزند تعریف کنیم که برابر message از کلاس والد باشد و بعد این پارامتر جدید را با دسترسی public این بار به کلاس Program بیاوریم:

```
public static string message2 = Girl.message;
```

همانطور که می بینید در کلاس **Girl** عبارت Girl.message به متغیر message در کلاس Father بدون هیچ ایرادی دسترسی دارد و مقدار خودش را به متغیر جدید message2 نسبت داده است. حالا کافیست برای نمایش این بار در کلاس Program متغیر **Girl.message2** را صدا بزنیم.

ذخیره و بازیابی

فهم این بخش بسیار مهم است. بعد از محاسبات که شخصیت کامپیوتر را تعریف می کند، بیشترین کاربرد این دستگاه در مراکز کاری و سازمان ها به ذخیره ی اطلاعات و داده های خام و سپس روش های بازیابی یا دریافت مجدد آنها بر میگردد. در واقع قرار نیست هر بار که اطلاعاتی را به کامپیوتر می دهیم، از بین بروند و برای دفعات بعد مجبور به ورود مجدد آنها باشیم.

به همین دلیل در همه ی زبانهای برنامه نویسی امکانات ویژه ای برای خود آن زبان وجود دارد که فارغ از اتصال به بانک های اطلاعات (Database) می توان عمل ذخیره و بازیابی را کدنویسی کرد. برای هر برنامه نویسی ضروری است که از این امکانات باخبر باشد. موضوعی که در بیشتر دوره های آموزشی از آن غفلت شده و فقط به طرح روش اتصال به بانک های اطلاعاتی بسنده می کنند.

در سی شارپ دو کلاس داخلی تعریف شده که یکی به نام StreamWriter وظیفه ی نوشتن در فایل ها را بر عهده می گیرد و دیگری StreamReader کارش خواندن محتوا از این فایل های متنی است. همچنین در این کلاس های غیر استاتیک که لازم است از روی آنها نمونه سازی شود، عملگرهای مختلفی هم برای کار با فایل پیاده سازی شده اند:

```
using System;
using System.IO;

namespace console_test
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamWriter sw = new StreamWriter("message.txt");
            sw.WriteLine ("Hi friends!");
            sw.Close();
        }
    }
}
```

اولین توضیح برای استفاده از این کلاس ها آن است که در هسته ی اولیه ی سی شارپ نباید به دنبال چنین کاربردهای خاصی باشیم. یعنی با using کردن کتابخانه ی کد System نمی شود به تنهایی به همه ی دستورات طبقه بندی شده ی سی شارپ دسترسی داشت. بعدها هنگام کار در پروژه های پیشرفته تر با کتابخانه های بیشتری آشنا خواهید شد. مثل کتابخانه ای با نام System.Globalization که وظیفه اش افزودن توابع جهانی و مواردی مثل تقویم فارسی به این زبان است! حالا در اینجا همانطور که در خط دوم برنامه دیده اید، اول برای کار با عملگرها و کلاس های مربوط به ورودی یا خروجی اطلاعات، کتابخانه ای System.IO را (Input/Output) به برنامه ی خود اضافه کنیم تا امکانات آن دستورات و خودتکمیلی کدها برای ما در محیط ویرایش کد آشکار شود.

توضیح باقی کدها ساده است. از روی کلاس StreamWriter درون متغیر sw یک نمونه ی جدید میسازیم و در همان مسیر پروژه فایل message.txt را معرفی می کنیم تا محتوای مورد نظرمان که می تواند رشته های ثابت یا مقادیر درون متغیرها باشد، داخل این فایل ذخیره شوند.

در خط بعد با استفاده از عملگر WriteLine. این بار از روی نمونه ی ساخته شده یعنی sw دستور ذخیره درون فایل را می دهیم تا عبارت Hi friends! در آنجا نوشته شود. در آخر نیز ضروری است که کل فرآیند نگارش در فایل با عملگر Close. به اتمام رسیده و فایل بسته شود و گرنه ذخیره انجام نمی شود. با همین فرآیند ساده اگر برنامه را اجرا کنید خواهید دید که فایل

message.txt در محل پروژه ایجاد شده و اگر آن را با نوتپد یا خود وی اس کد باز کنید، درونش محتوای ذکر شده را می بینید.

به روش بهینه تر این کار توجه کنید:

```
static void Main(string[] args)
{
    using (StreamWriter sw = new StreamWriter("message.txt"))
    {
        sw.WriteLine("Hi Doostan!");
    }

}
```

این بار با اجرای برنامه متوجه می شوید که مقدار جدید یعنی Hi Doostan! جایگزین مقدار قبلی در فایل شده و البته به دلیل استفاده از بلوک using دیگر نیازی به استفاده از عملگر Close. برای بستن فایل و خالی کردن حافظه نیست. در واقع دستور using در وسط برنامه به طور خودکار فرآیند مدیریت حافظه را هنگام نمونه سازی از روی کلاس ها به عهده گرفته و به محض بسته شدن بریس یا بلوک های آن آزادسازی حافظه صورت می گیرد. به همین دلیل استفاده از این روش نیز توسط خود مایکروسافت توصیه شده است.

حالا وقت آن رسیده که روش خواندن از درون فایل را هم تست کنیم. در واقع قرار است که این استفاده از فایل کامل باشد. یعنی گاهی مقادیر ورودی برنامه از فایل باشد:

```
string message;
using (StreamWriter sw = new StreamWriter("message.txt"))
{
    sw.WriteLine("Hi Doostan!");
}
using (StreamReader sr = new StreamReader("message.txt"))
{
    message = sr.ReadLine();
}
Console.WriteLine(message);
```

```
Console.ReadKey();
```

روشن است که ReadLine را هم می توان مانند کلاس WriteLine بدون using نوشت. ولی این روش بهینه است. در اینجا هم یک نمونه از روی آن ساخته و به متغیر دلخواهی مثل sr نسبت می دهیم تا عملگرهای آن را راحت استفاده کنیم. با ReadLine. خط اول درون فایل را خوانده و به متغیر message نسبت می دهیم و در نهایت آن را در کنسول نمایش می دهیم.

تا اینجا فرآیند نوشتن و خواندن یک خط را درون فایل انجام دادیم. اگر بخواهیم مجموعه ای از خطوط و اطلاعات در فایل وارد کرده و بخوانیم، همانطور که حدس می زنید باید از حلقه ها بهره ببریم:

```
string fileName = @".\message.txt"; // : تعریف کلی فایل برای عملیات
        // ------ : بخش نوشتن مقدار در فایل -------
try     // : چک کردن عملیات برای انجام در صورت عدم بروز خطا
{
    using (StreamWriter sw = new StreamWriter(fileName, true))
    {
        //sw.WriteLine("Sample text1." , Environment.NewLine);
        sw.Write("Sample text." + "\n");  // : نوشتن محتوا در فایل
    }
}
catch (Exception ex)  // : نمایش خطا در صورت بروز هنگام نوشتن در فایل
{
    Console.Write(ex.Message);
}
        // ------ : بخش خواندن مقادیر از فایل -------
try
{
    using (StreamReader sr = new StreamReader(fileName))
    {
        string line;  // : تعریف متغیر برای خواندن هر خط در فایل
        while ((line = sr.ReadLine()) != null)  // حلقه تا زمان پوچ نبودن خط
        {
            Console.WriteLine(line); // : نمایش خط به خط فایل
        }
```

```
    }
}
catch (Exception exp)
{
    Console.WriteLine(exp.Message);
}
Console.ReadKey();
```

این برنامه کامل ترین حالت ممکن برای نوشتن و خواندن در فایل متنی است. کامنت ها به خوبی هر بخش را توضیح می دهند. چند بار این برنامه را اجرا کنید. خواهید دید که هر بار برنامه را اجرا کنید، یک خط جدید داخل فایل اضافه می شود. دلیل این امر در فعال کردن بخش append یا افزودن به مقادیر قبلی در کلاس StreamWriter است. یعنی همان true که مقابل filename آمده است.

اما دو حالت نوشتن در فایل را می بینید که نوع اول آن به صورت کامنت در آمده است. در واقع نوع اول و دوم هر دو یک جور عمل می کنند. یعنی بعد از نوشتن یک خط فایل را آماده می کنند تا در خط بعد مقدار جدید وارد شود. همینطور \n بعد از عملگر Write یعنی برو خط بعد.

در نهایت هم حلقه ای داریم که خط به خط محتوای درون فایل را می خواند.

به مجموعه ی چهار فرآیند

نوشتن Create، **خواندن** Read، **ویرایش** Update، و **حذف** Delete

به طور اختصار **CRUD** می گویند.

به طور معمول فرآیندهای کار با محتوا در فایل در دستورهای مربوط به بانک های اطلاعاتی به سادگی انجام می شوند. اما در اینجا در سی شارپ تنها و مستقل بدون فراخوانی بانک های اطلاعاتی، از کلاس های ویژه ی خود سی شارپ استفاده می کنیم.

کلاس **File** در سی شارپ، مجموعه ای از اقدامات جالبی را روی فایل ها انجام می دهد. ابتدا سعی می کنیم در ارتباط با مطلب قبلی یعنی خواندن و نوشتن محتوا در فایل متنی، ویژگی هایی مثل حذف یا ویرایش را هم بررسی کنیم:

```
// ------ بخش ویرایش : -------
```

```
string text = File.ReadAllText("message.txt");
text = text.Replace(".", string.Empty);
//text = text.Trim();
File.WriteAllText("message.txt", text);
```

اضافه کردن این بخش به انتهای برنامه ی قبلی یا اجرای آن در غالب یک برنامه ی جدید هم مقدور است. عملگر **ReadAllText.** از کلاس File تمام متن فایل را می خواند. بعد از اینکه این متن و نمونه ی ساخته شده از کلاس و عملگر را به متغیر text نسبت دادیم، با عملگر **Replace.** می توانیم دو مقدار را با هم جابجا کنیم. در این مثال "." را با محتوای خالی یعنی **string.Empty** عوض می کنیم. این کار در همه جای فایل به هر تعداد رخ می دهد. می توانیم به جای مقدار دوم هم باز یک رشته حروف جدید قرار دهیم تا جایگزین شوند. خط آخر هم تغییرات را کامل کرده و انجام می دهد.

بخش Trim. که کامنت شده مقادیر خالی و اسپیس های ابتدا و انتهای فایل را می تواند حذف کند. می توانید مجموعه ی عملگرهای مختلف دیگر را هم که با خاصیت خودتکمیلی در ویرایشگر کد قابل مشاهده است، تست کنید.

از تغییرات محتوای فایل متنی که بگذریم به کار روی خود فایل ها و حذف و کپی می رسیم که با چند مثال ساده می توان آنها را یاد گرفت:

```
System.IO.File.Copy("c:\\message.txt", "d:\\copy.txt");
```

مثال بالا فایل message.txt را از درایو C با نام جدید copy.txt در درایو D کپی می کند.

```
System.IO.File.Copy("c:\\message.txt", "d:\\copy.txt", true);
```

اضافه کردن پارامتر true در انتها باعث می شود که در صورت وجود فایل، عمل Over write یا بازنویسی روی فایل موجود، انجام شود.

```
System.IO.File.Move("c:\\message.txt", "d:\\copy.txt");
```

اگر به جای Copy. از Move. استفاده کنید، فایل از مبدأ خودش پاک شده و به جای جدید منتقل می شود.

```
if (System.IO.File.Exists("d:\\message.txt"))
{
    // کد مورد نظر شما
}
```

در مثال بالا وجود فایل را در مسیر مشخص شده می توانیم چک کرده و عملیات مورد نظر را پی بگیریم.

تا این لحظه با مطالب مطرح شده، کلیات زبان برنامه نویسی سی شارپ را شناخته ایم. همانطور که برای صحبت کردن به زبان انگلیسی نیاز نداریم که تمام فرهنگ لغات یا دیکشنری آن زبان را حفظ باشیم، در اینجا نیز مجموعه ای از عملیات رایج در برنامه نویسی را که در زبان سی شارپ برای ساخت برنامه به طور مستقل لازم است، شناخته ایم. از این به بعد کافیست بسترهای مختلف اجرای برنامه اعم از ویندوز (دسکتاپ)، برنامه ی موبایلی (آندروید یا اپل) و اپلیکیشن تحت وب را برای ادامه انتخاب کنیم. حالا به راحتی می توانیم یادگیری هر کدام از این محیط ها را با خیال راحت شروع کنیم:

- **ASP.net** : بهترین محیط برای ساخت نرم افزارهای تحت وب یا Web applications است که حالا دیگر به راحتی در محیط لینوکس هم قابل اجرا و بهره برداری است. خروجی این برنامه ها روی تمام مرورگرهای اینترنتی یا Browser در هر سیستم عاملی حتا موبایل یا تلوزیون های هوشمند هم قابل مشاهده است.
- **Xamarin** : با تلفظ زامارین محیطی را برای کدنویسی سمت موبایل فراهم می کند. به راحتی یک بار توسط کدهای سی شارپ برنامه نوشته و بعد روی تمام محیط های سمت موبایل مانند Android یا iOs و همچنین Windows mobile برنامه های شما اجرا می شوند.
- **Unity** : محیط منحصر به فرد تولید بازی و انواع Game به شکل های دو بعدی و تخت یا سه بُعدی که هنگام خروجی گرفتن می

توان تعیین کرد به راحتی برای چه محیطی از وب، موبایل یا حتا کنسول های بازی مثل xBox و Play station کار نهایی ساخته شود.

حتا در هر یک از زمینه های کاری فوق شما انواع خدمات و سرویس ها را می توانید مثل سرویس های ابری تحت اینترنت نوشته و به کار بگیرید. به طور معمول اما چیزی که نیاز بازار کار است، عنوانی کلی به نام نرم افزار تحت وب است. وب اپلیکیشن می تواند خدمات اطلاع رسانی متعددی را از بستر وب و شبکه ارائه کند که روی هر کامپیوتر یا موبایل هوشمندی هم بدون تغییر خاصی قابل دریافت بوده و نیاز به برنامه نویسی ویژه ندارد.

حتا ظهور نوع خاصی از اپلیکیشن های تحت وب به نام PWA که مخفف Progressive web app هستند کمک می کند که حالت های واکنش گرا (Responsive) از برنامه های تحت وب را که در موبایل ها به صورت خاص نمایش داده می شوند، بدون اجرای مرورگرها و منوهای مزاحم آنها، به صورت یک اپ مستقل در هر سیستم عامل ذخیره و با آیکون مشخص اجرا کنید.

اپلیکیشن تحت وب

از مزایای مهم برنامه ی تحت وب می توان به بی نیاز بودن آن برای نصب و اجرا روی سیستم عامل خاصی اشاره کرد. همین موضوع از سرایت ویروس های کامپیوتری جلوگیری می کند و احتمال هک را از بین می برد که مزیت مهمی است. وب اپلیکیشن ها به این دلیل که روی مرورگرهای اینترنتی اجرا می شوند دیگر وابسته به سیستم عامل خاصی برای اجرا نیستند.

بی هیچ مقدمه ای با توجه به مطالبی که در بخش اول یعنی کنسول فهمیده ایم می توانیم نمونه ی تحت وب را هم ایجاد کنیم. یک پوشه مثلاً به نام my-site روی دسکتاپ ایجاد کرده و بعد از باز کردن خط فرمان در مسیر آن پوشه به سادگی دستور زیر را اجرا کنید:

```
dotnet new webapp
```

همین! هر چند که **--help** اینجا هم بعد از این سه کلمه می تواند انواع آپشن و انتخاب های متنوعی را برای ایجاد پروژه ی خام وب معرفی کند. باز هم با **code .** ویرایشگر کد را در مسیر پروژه باز کنید.

اینجا با اولین فرق اساسی وب اپلیکیشن با پروژه ی کنسولی مواجه می شوید. درون Program.cs چند خطی دستور ناشناس دیده می شود که ظاهر آنها حکایت از اجرای یک Host یا میزبان اینترنتی دارد. بله پروژه ی تحت وب که قرار است خروجی آن روی مرورگر نمایش داده شود، نیاز به یک خدمات دهنده یا سِرور وب دارد. قبل از هر توضیح گیج کننده و دلسرد کننده ای برای اینکه متوجه ی جذاب تر بودن کار تحت وب شویم، این پروژه ی خام را به همان سادگی پروژه ی کنسول اجرا کنید:

dotnet run

و نتیجه اگر همه چیز درست پیش برود در خط فرمان به صورت زیر خواهد بود:

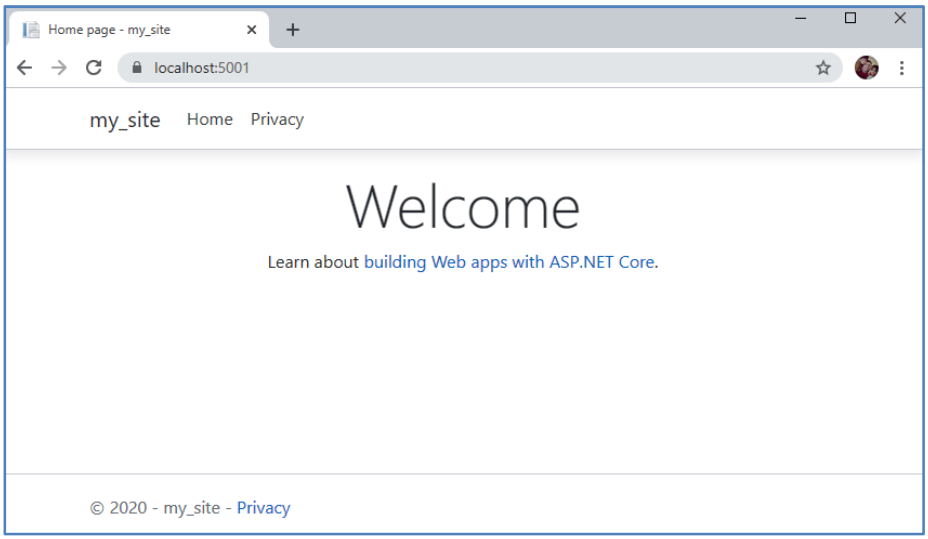
C:\Users\Pedram Rahimi\Desktop\my-site> **dotnet run**
Building...
info: Microsoft.Hosting.Lifetime[0]
Now listening on: **https://localhost:5001**
info: Microsoft.Hosting.Lifetime[0]
Now listening on: **http://localhost:5000**
info: Microsoft.Hosting.Lifetime[0]
Application started. **Press Ctrl+C to shut down.**
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: **Development**
info: Microsoft.Hosting.Lifetime[0]
Content root path: C:\Users\Pedram Rahimi\Desktop\#

بله این خروجی همان چیزی است که در محیط کنسول انتظار داریم. چون خروجی تحت وب را قرار است که روی مرورگر ببینیم. سِرور تحت وب ما که در نسخه های قدیمی دات نت به آسانی قابل تنظیم و اجرا نبود، حالا بخاطر پیشرفت های نسخه ی اوپن سورس با یک دستور ساده در دسترس است و دیگر نیازی هم به راه اندازی برنامه ی مجزا مثل IIS که مخفف Internet information service است، نداریم. بی هیچ مقدمه ی دیگری به سراغ مرورگر اینترنتی خودتان رفته و نشانی زیر را در آن وارد کنید:

http://localhost:5000

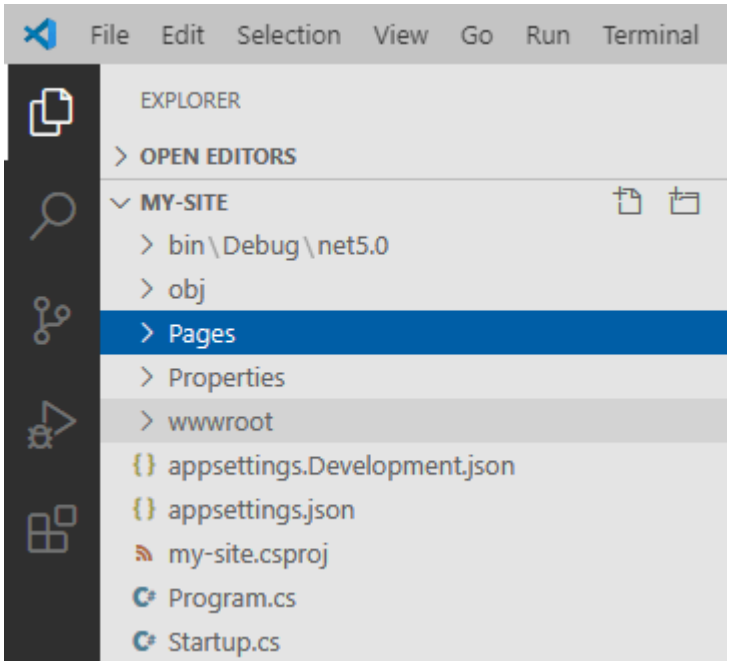
با توجه به اینکه پروژه ی تحت وب شما به طور خودکار روی محیط امن یا https هم تنظیم شده، به طور خودکار به نشانی localhost:5001 ارجاع داده می شود. چنین جزئیات و اطلاعاتی را در مبانی شبکه و دوره ی Network+ (با تلفظ نت ورک پلاس) می توان یاد گرفت که ربطی به فراگیری برنامه نویسی و سی شارپ ندارند. اما در همین حد که بدانیم پروژه ی تحت وب ما هنگام کدنویسی روی نشانی وب محلی یا همان لوکال هاست اجرا شده و آی پی (IP) یا نشانی منحصر به فرد آن از نظر علم شبکه در اصل 127.0.0.1 است، کفایت می کند. این آدرس از هیچ کجا خارج از کامپیوتر ما قابل دسترس نیست و همیشه به دستگاهی اشاره می کند که در حال کار با آن هستیم. به محض قرار دادن پروژه روی سِرورهای وب اینترنتی یا شبکه های سازمانی، برنامه ی ما با نشانی آن مکانها بی نیاز از هر تغییری و با آن نشانی جدید در دسترس خواهد بود.

نتیجه ی اجرا به صورت زیر است:



این شاید یک جور Hello world در مقیاس وب و البته همراه کتابخانه و پوسته ی ویژه (Template) مایکروسافت برای ایجاد سریع ترین نرم افزارهای تحت وب جهان با نام Razor است (تلفظ آن Reyzer است). اگر به خاطر داشته باشید اجرای دستور dotnet new به تنهایی در خط فرمان، فهرستی از تمام پروژه های قابل ایجاد را در اختیار شما می گذارد که ریزر یکی از آنهاست.

حالا به محیط VScode می رویم تا ساختار پروژه ی تحت وب را بشناسیم. تا اینجا فقط فهمیدیم که مانند کنسول در Program.cs کاری قرار نیست که انجام دهیم. ساختار فایل های پروژه را نگاه کنید:



همانطور که متوجه شده اید این کتاب با طرح تمام جزئیات پروژه های سی شارپ باعث خستگی و دلزدگی فراگیران نمی شود. به همین دلیل در اینجا هم توصیه می شود که بدون کنجکاوی خاصی، تنها در نظر داشته باشید که فعلاً روی دو پوشه ی **Pages** و **wwwroot** کار خواهید کرد و نیازی به سر زدن به جاهای دیگر ندارید.

یکی از ویژگی های منحصر به فرد این کتابخانه ی کد یعنی Razor که با تمپلیت آماده و ساده اش ساخت برنامه ی تحت وب را از هر محیط دیگری ساده تر می کند، بحث امنیت آن است. به

عنوان یک نمونه ی ساده که کاربرد پوشه ی wwwroot را هم نشان می دهد، یک فایل معمولی وب را با فرمت html. ساخته و در این پوشه قرار دهید. مثلاً با نام **test.html** که متن ساده ای را هم درون آن نوشته باشید.

نکته: ایجاد سریع یک صفحه ی HTML

در محیط وی اس کد، یک فایل جدید با پسوند html. مثلاً **test.html** بسازید. در خط اول آن تایپ کنید **doc** و کلید **Tab** را بزنید. اگر افزونه های ویژه ی HTML درست کار کنند.، نتیجه درج سریع تگ های تکراری و ضروری HTML خواهد بود و شما می توانید متن مورد نظر خود را فقط بین تگ های باز و بسته ی body قرار دهید:

```
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title> Document </title>
</head>
<body>
   سلامی چو بوی خوش آشنایی!
</body>
</html>
```

حالا می توانید این صفحه را به راحتی در پروژه ی سایت خود با نشانی زیر فراخوانی کنید:

https://localhost:5001/test.html

حتا می توانید در wwwroot که در اصل روت یا ریشه ی سایت شما را تشکیل می دهد، پوشه های مختلفی ساخته و فایل ها یا صفحات ایستا را در آنجا طبقه بندی کنید و البته برای صدا زدن آنها باید مسیر پوشه ها را نیز در آدرس مرورگر بیاورید.

صفحه ی سی شارپی

باید بدانید که خارج از این پوشه ی ریشه یا همان wwwroot هیچ فایل دیگری از پروژه هنگام فراخوانی از نوارآدرس مرورگر، دردسترس نیست و این موضوع به خوبی امنیت فایل ها را در نرم افزار تحت وب ما نشان می دهد.

به عنوان مثال صفحات پویا (Dynamic) و حاوی کدهای سی شارپ که درون پوشه ی Pages قرار دارند، به طور مستقیم در دسترس کاربران از روی وب قرار نمی گیرند، بلکه نتیجه ی اجرای آنهاست که به مرورگر ارسال می شود. برای فهم روش این کار در پوشه ی **Pages** فایل **index.cshtml** را باز کرده و کدهای آن را بررسی می کنیم. فایلهایی با پسوند cshtml با ترکیبی از سی شارپ و اچ تی ام ال، مکانی هستند که قرار است دستورات سی شارپ خود را برای اجرا در پروژه ی تحت وب بنویسیم (نه مانند پروژه های کنسول که در Program.cs می نوشتیم). طبق روال چند تایی از بخش های این صفحه ی نمونه را توضیح داده و تغییر می دهیم:

```
@page   // دستور پیج یعنی این صفحه قابل فراخوانی در نوارآدرس مرورگر باشد
@model IndexModel  // کلاسی به نام "ایندکس مدل" برای کدهای سی شارپی این صفحه  است
@* برای کامنت گذاشتن می توان از یک ادساین و ستاره هم استفاده کرد *@
@* برای ورود کدهای سی شارپی می توان یک بلوک که با ادساین شروع شده مثل زیر ساخت  *@
@{
    // عنوان صفحه را در مرورگر می توان به صورت زیر تغییر داد:
    ViewData["Title"] = "خانه";
    //  یک متغیر جدید را به صورت زیر می توان ساخت:
    string message = "دوستان سلام";
}
```

```
<div class="text-center">
    <h1 class="display-4">  خوش آمدید  </h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps w
ith ASP.NET Core</a>.</p>
</div>
@* مقدار متغیر ساخته شده را می توان لای اچ تی ام ال با یک ادساین نمایش داد:  *@
@message
```

خط اول یعنی دستور @page یکی از موارد لازم برای تبدیل شدن این فایل به صفحه ی آدرس دار سایت است. همیشه پروژه ی دات نت به دنبال فایلی با نام index.cshtml می گردد تا اولین صفحه ی سایت قرارش داده و آن را نمایش دهد.

با هم اول **مراحل ساخت یک صفحه** یا پیج جدید را به ترتیب زیر انجام می دهیم:

- در پوشه ی Pages یک فایل جدید مثلاً با نام Test.cshtml بسازید.

نکته ی با اهمیت آن است که حرف اول این فایل حتماً باید با حروف بزرگ انگلیسی باشد.

- روی فایل جدید Test.cshtml از پنل سمت چپ وی اس کد کلیک کرده و آن را باز می کنیم تا عبارات زیر را در آن وارد کنیم:

```
@page
من صفحه ی تست هستم
```

- با CTRL+S فایل را ذخیره کنید.
- اگر پروژه در حال اجرا است، آن را با **CTRL+C** درون خط فرمان، متوقف کنید و دوباره با دستور dotnet run اجرایش کنید. به این ترتیب پروژه مجدد مراحل بیلد را طی می کند.
- در نوار آدرس مرورگر نشانی زیر را وارد کرده و اینتر بزنید:

https://localhost:5001/test

به همین راحتی یک پیج جدید به پروژه اضافه و با نام فایل بدون هیچ پسوندی در آدرس وبی فراخوانی میشود. این کار برای فهم اولین دستور در پیج ایندکس بود. حالا دوباره به سراغ صفحه ی اول سایت یعنی index.cshtml می رویم. راستی نشانی زیر را هم تست کنید:

https://localhost:5001/index

بله index یا پیج پیشفرض را می توان در آدرس ننوشت. به طور خودکار این آدرس با ریشه ی سایت یک چیز فرض می شود.

اما همانطور که متوجه شده اید صفحه ی Index.**cshtml** در کنار خودش یک فایل جانبی به صورت Index.**cshtml.cs** دارد. در واقع این فایل یک کلاس است که صفحه ی مرتبطش از آن ارث بری می کند. این کار برای این انجام شده که کدنویسی به طور کامل مرتب و تمیز انجام شود و بتوان عملگرهای طولانی و کدهای پیچیده تر را به طور کامل در این کلاس یا مدل تعریف کرد و لابلای HTML ننوشت.

در واقع دستور دوم در صفحه یعنی **@model** اشاره به همین فایل جانبی دارد.

اما با توجه به کدهایی که در محتوای این کلاس مشاهده می کنید، ساخت دستی آن نمی تواند روشی منطقی باشد. بنابراین یک بار دیگر صفحه ای را این بار با نام خودمان می سازیم. البته با روش اصولی که مدل آن نیز ساخته شود:

- در خط فرمان و مسیر پوشه ی پروژه ی تحت وب خودمان با دستور زیر وارد پوشه ی Pages می شویم:

```
cd pages
```

- حالا دستور زیر را برای ساخت صفحه و مدل مربوطش به نام خودمان یا هر نام دیگر وارد می کنیم. البته باید حرف اول صفحه بزرگ باشد:

```
C:\..\my-site\pages> dotnet new page --name Pedram
```

خواهید دید که صفحه ی Pedram.**cshtml** و Pedram.**cshtml.cs** هر دو به طور خودکار در مسیر Pages ساخته می شوند. در واقع از این به بعد صفحات خود را به این شکل ساخته و با کدهای هر دو بخش کار خواهیم کرد.

حالا برای اینکه بدانیم چطور می شود در لابلای کدهای HTML درون هر صفحه ی cshtml ، دستورات سی شارپ نوشت، به الگوی زیر توجه کنید:

```
@{
    // محل قرارگیری کدهای سی شارپ در صفحات ریزر
}
```

به هر تعداد که نیاز باشد می توانید از این بلوک ها که با علامت @ (اد ساین) در ابتدای آن شروع شده، استفاده کنید. در پایان کار و هنگام بیلد شدن پروژه، تمام این بلوک ها به صورت یک کلاس واحد و همراه مدلی که مرتبط با این صفحه است، تحلیل شده و کار خواهند کرد.

همانطور که در نمونه می بینید، متغیر رشته ای جدیدی هم به نام **message** تعریف کرده ایم که مقدار فارسی "دوستان سلام" را به آن داده ایم. یعنی بر خلاف پروژه های کنسولی، ما در وب می توانیم به هر زبان و فرهنگی که مایل باشیم داده های خود را نمایش دهیم.

برای نمایش این پیام در خط آخر هم از الگوی زیر پیروی کرده ایم:

نام متغیر، عملگرها یا کلاس هایی که خروجی دارند برای نمایش مقدار @

همانطور که روشن است اینجا خبری از **{}** نیست و عملکرد @ متفاوت است. یعنی لای تگ های HTML مقادیر متغیرها را به محض رسیدن به این خط نمایش می دهد. درون @{ } هم می توان از این حالت مابین دستورات سی شارپی استفاده کرد که فقط باید انتهای آن ; قرار داد.

همچنین اگر دستورات سی شارپی شما فقط یک خط باشند هم می توان بدون { } آنها را طوری نوشت که ابتدای هر خط یک @ آمده باشد. مثلاً:

```
<ul>
@for (int i = 0; i < 10; i++) {
<li>@i</li>
}
</ul>
```

اگر این خطوط را به انتهای یکی از صفحات اضافه کرده و پروژه را مجدداً اجرا کنید، می بینید که به زیبایی اعداد ۰ تا ۹ را در قالب یک لیست نقطه دار برای شما نمایش می دهد.

در یک آزمایش جالب دیگر با توجه به آموخته های بخش کنسول می توانیم محتوای یک فایل متنی را هم درون صفحه ی وب خود فراخوانی کرده و به این ترتیب یک محتوای داینامیک هم داشته باشیم. برای این منظور ابتدا در ریشه ی پروژه ی خود (نه در پوشه ی Pages بلکه یک پوشه بیرون از آن) فایل **message.txt** را کنار Program.cs ساخته و یک محتوای دلخواه در آن وارد کنید. حالا خطوط زیر را در انتهای صفحه ی اول یا هر صفحه ی دیگر وارد کنید:

```
@using System.IO;
@using (StreamReader sr = new StreamReader("./message.txt"))
{
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        @line <br>
    }
}
```

این دستورات در واقع مربوط به پروژه های قبلی ما در بخش کنسول برای خواندن از فایل متنی هستند که عیناً در اینجا وارد شده اند. اما تنها کمی تفاوت های جزئی برای استفاده در صفحات Razor پیدا کرده اند. همانطور که معلوم است، ابتدای دستورات @ قرار گرفته و درون حلقه نیز برای نمایش مقدار هر خط بجای Console.Write یا دستورات مربوط به کنسول، باز از همان @ استفاده شده.

البته به زیبایی یک تگ
 هم برای شکستن و رفتن به خط بعد قابل استفاده است. این ویژگی یعنی استفاده ی آسان تگ های HTML کنار دستورات زبان سمت سِرور، در هیچ نمونه ی دیگری غیر از سی شارپ دیده نمی شود. به طور مثال در PHP که یک زبان تفسیری و غیرکمپایلی یا بدون بیلد شدن است، حتماً باید محدوده ی دستورات PHP از HTML تفکیک شده باشند و برای چاپ تگ ها کنار دستورات باید آنها را درون کوتیشن و نظایر آن قرار داد.

تا این لحظه به راحتی می توانید سایت هایی را ایجاد کنید که به صورت پویا مطالبی را از درون فایل های متنی خوانده و در صفحات مختلف سایت به نمایش بگذارند. همچنین محاسباتی را درون این صفحات انجام داده و نتیجه را نمایش دهید.

برای کنجکاوی بیشتر در پایان این بخش، می توانید فایل **_layout.cshtml** را در مسیر **Pages/Shared** باز کرده و تغییر دهید. (دلیل این شکل نامگذاری در فایل مذکور که یک آندرلاین ابتدای آن آمده، این است که چنین فایل هایی مستقیم در پروژه اجرا نمی شوند بلکه اشتراک گذاشته شده و در دل فایل ها و صفحات صدا زده می شوند). مثلاً با توجه به آنچه که خارج از دوره ی آموزش سی شارپ و در مبانی وب یا HTML ممکن است فهمیده باشید، پوسته ی سایت خود را از این قسمت تغییر دهید. حتا یک تغییر کوچک هم در قیافه ی برنامه باعث می شود که احساس کنید کل این پروژه و ساختار کدهای آن را در اختیار دارید. به عنوان مثال تگ بدنه را به صورت زیر تغییر دهید:

```
<body dir="rtl">
```

بله بعد از اجرای مجدد پروژه خواهید دید که منوهای شما **راستچین** (**RTL**:Right to Left) و مناسب برای زبان فارسی شده اند! همچنین در هر کدام از صفحات و پیج های سی شارپی خود مثل Index.chtml هم می توانید با قرار دادن محتوای صفحات بین یک تگ div با مشخصات زیر، آن محتوا را هم راستچین کنید:

```
<div align="right">
    محل قرار گیری محتوای قابل نمایش صفحات
</div>
```

در مرحله ی بعد باید بتوانیم ارتباطی بین اشیاء قابل تعریف در صفحات HTML با دستورات سی شارپ برقرار کرده و کنترل آنها را به دست بگیریم.

کار با اشیاء وب

زبان HTML تنها برای ساختن ظاهر صفحات وب بوده و زبان برنامه نویسی نیست. هیچ کنترل مؤثری روی اشیاء در صفحه ی وب نداریم مگر آنکه از زبان دومی که خاص برنامه نویسی است استفاده کنیم. کتابخانه های کد که برای کار تحت وب ساخته می شوند هم به خوبی می توانند با اشیاء وب که از طریق زبان HTML به وجود آمده اند ارتباط برقرار کنند. قبل از هر چیز به کدهایی که دو شیء مهم وبی را می سازند نظری می اندازیم:

<input type=”text” >

<button asp-page=” “ > ارسال </button>

این دو شکل بسیار کاربردی تر از بقیه هستند. اگر کنجکاو به فهم روش ساخت شکل های دیگری هستید یک جستجوی ساده در مورد تگ یا برچسب های HTML در اینترنت کافیست. البته با چیزی که از مبانی وب یاد خواهید گرفت می دانید که فرم های ورود اطلاعات برای ارسال، درون یک تگ ویژه به نام **form** قرار گرفته و برای اینکه هر شیء را هم بتوان به طور مستقل شناخت، می توان یک پارامتر به اسم name را درون این اشیاء مقداردهی کرد:

```
<form method="post">
   <input type="text" name="testText">
   <button asp-page="./pedram"> ارسال </button>
</form>
```

همچنین پارامتر method می تواند از دو نوع اصلی در فرم ارسال اطلاعت پیروی کند. یکی GET که مقادیر ارسالی فرم را بدون لحاظ کردن ضرایب امنیتی و از طریق آدرس می فرستد. دیگری POST که داده ها را مخفی و کدشده می فرستد. بالطبع برای مواردی مثل ارسال مقادیر سرچ و جستجو get مناسب است. اما برای مقادیری که نیاز به ذخیره ی آنها در دیتابیس داشته باشیم post را انتخاب می کنیم تا در خطر دستبرد هکرها نباشد.

پارامتر asp-page هم مقصد را تعریف می کند. جایی که فرم به آن ارسال می شود. اگر این پارامتر تعیین نشود، فرم برای صفحه ی خودش یعنی جایی که این دستورات فرم نوشته شده، ارسال میشود.

زمان آزمایش فرا رسیده. ما در وب بر خلاف کنسول که به صوت خطی داده ای را از کاربر گرفته و روی آن عملیات مورد نظر خود را انجام می دادیم، فرمی داریم که وقوع رویدادهای مختلف آن از پُر کردن جعبه های متن تا فشردن کلید ارسال نامعلوم است. در این صورت عملگر یا توابعی داریم که این تغییرات را رصد می کنند.

- مثلاً به سراغ فایل **Privacy.cshtml.cs** می رویم. در این فایل مدل که کدهای صفحه ی Privacy.cshtml را در خود نگه می دارد کدهای زیر را در ادامه ی کدهای کلاس PrivacyModel وارد کنید:

```
public void OnGet()
{
}
public string testText   { get;  set;  }
public void OnPost()
{
    testText = Request.Form["testText"];
}
```

عملگر onGet برای کنترل فرم های ارسالی از طریق پروتکل GET است. که به صورت پیشفرض در مدلها ساخته می شود. ما برای اینکه فرم خود را با POST ارسال کنیم، عملگر onPost را نوشته ایم. متغیر رشته ای testText را به صورت یک ویژگی یا Property از این کلاس تعریف کرده ایم.

get و set بخش هایی هستند که به طور پیشفرض تعریف نمی شوند. فعلاً در همین حد بدانید که get برای کنترل زمانی است که از متغیر مربوط به خودش مقداری فراخوانی یا خوانده می شود و set مخصوص کنترل زمانی است که به این متغیر یا پراپرتی مقدار جدید داده بشود. در این صورت در مقابل هر کدام از get و set یک { } باز شده و کد عملیات مورد نظر برای کنترل زمان مقدار دهی یا خواندن مقادیر نوشته می شود.

برای ساخت خودکار یک پراپرتی کافیست عبارت **prop** را در ویرایشگر VScode وارد کرده و کلید Tab را بزنید. بعد با فشار مجدد تب بین بخش های مختلف دستور نوشته شده حرکت کرده و آن را تغییر دهید.

عبارت `Request.Form["testText"]` در واقع به مقدار همان عنصری از Form اشاره دارد که با `name="testText"` معرفی شده. بنابراین به محض ارسال فرم و فراخوانی عملگر onPost مقدار value از آن تکست باکس را دریافت کرده و به متغیر testText نسبت می دهد. حتا می توان برای درک بهتر این بخش، نام متغیر را متفاوت از شیء تکست باکس در نظر گرفت.

در بخش قبل روش تعریف متغیر و فراخوانی آن را در فایل cshtml که بین `@{ }` انجام می شد، و با @ هم مقدار آن در صفحه به نمایش در می آمد، آشنا شدیم. اما حالا متغیری داریم که در صفحه ی cshtml.cs یا همان مدل تعریف شده. برای فراخوانی آن در cshtml که اینجا فایل **Privacy.cshtml** است، داریم:

```
<form method="post">
   <input type="text" name="testText">
   <button> ارسال </button>
</form>
<b>@Model.testText</b>
```

در هر مرحله از تغییرات فراموش نکنید که پروژه باید مجدد اجرا شود. همانطور که می بینید برای نمایش مقادیر ارسال شده توسط فرم که در مدل مرتبط با صفحه تعریف شده اند، کلمه ی **Model** در ابتدای آن ضروری است.

قابلیت ساخت فرم های محاسباتی هم به شما منتقل شد! فهم آن مشکل نیست. چون تکست باکس ها هم مقادیر پیشفرض رشته ای دارند، با توجه به چیزی که برای تبدیل رشته به عدد می دانیم، کافیست مثلاً قبل از جمع بستن مقادیر دو جعبه ی متن، آنها را به عدد تبدیل کنیم:

```
@(Convert.ToDouble(Model.testText) +
Convert.ToDouble(Model.testText2))
```

در این مثال فراموش نکنید که `testText2` را هم در فرم خود و هم پراپرتی های مدل، و هم عملگر کنترل ارسال فرم تعریف کرده باشید.

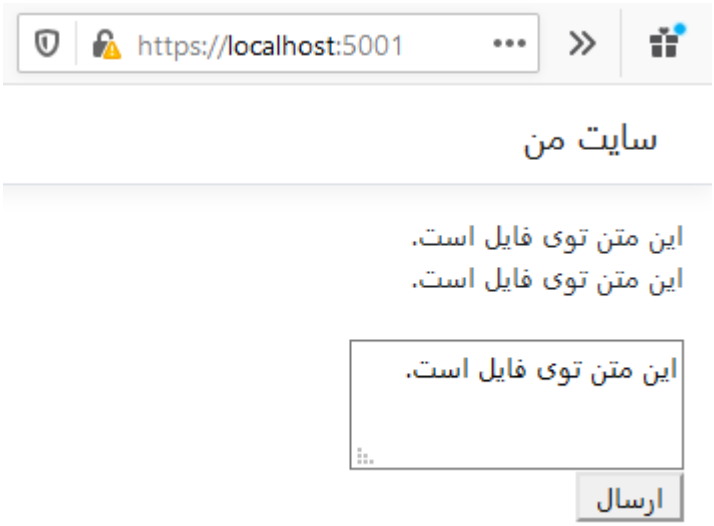
در همین جا ایده ی ساخت فرمی برای تغییر محتویات متنی صفحه را هم می توانید تحقق ببخشید! ابتدای کار با صفحات وب روش خواندن و نمایش متون از فایل را دیدید. کافیست این متون رو به یک شیء روی صفحه انتقال دهید. پس از ویرایش درون این شیء هم می توانید دستور ذخیره ی مجدد در فایل را صادر کنید:

```
            @*----- روش اول خواندن و نمایش از فایل -----*@
@using System.IO;
@using (StreamReader sr = new StreamReader("./message.txt"))
  {
     string line;
     while ((line = sr.ReadLine()) != null)
     {
        @line <br>
     }
  }
            @*----- روش دوم خواندن و نمایش از فایل -----*@
 @{
     string content = System.IO.File.ReadAllText("./message.txt");
  }
  @content <br>
          @*----- روش ساده ی ساخت فرم ورود اطلاعات -----*@
     <form method="post">
        <textarea name="textContent">@content</textarea> <br>
        <button>ارسال</button>
     </form>
```

خطوط بالا را در یکی از صفحات حتا **Index.cshtml** هم می توان وارد کرد. البته در اینصورت باید دکمه ی ارسال و تغییرات در محتوا را هم مدیریت کنیم. برای همین باید در فایل مدل یا در اینجا **Index.cshtml.cs** هم کدهای زیر را وارد کنید:

```
public void OnPost()
 {
     testText = Request.Form["textContent"];
     System.IO.File.WriteAllText("./message.txt", testText);
 }
```

تمام! همانطور که می بینید با هر بار تغییر در متن و فشردن دکمه ی ارسال، متن روی صفحه تغییر می کند:



البته همانطور که متوجه شده اید حتا زمانی که با روش دوم یعنی یک جا تمام فایل را می خوانید، امکان نمایش خط به خط آن نیست! یعنی اینترهایی که در آخر خطوط زده ایم لحاظ نمی شوند. همیشه بهترین روش برای نگهداری و بازخوانی متون آرایش شده، همان کدهای HTML است. اما به دلیل یکی دیگر از ویژگی های امنیتی دات نت، به طور پیشفرض امکان درج و بازخوانی این کدها غیر فعال شده. دلیل این امر امکان استفاده ی هکرها از تگ <script> و ذخیره و اجرای کدهای جاوااسکریپت در صفحات است که به آن حمله ی XSS گفته می شود (**C**ross **s**ite **s**cripting). با این حال ادیتورهای متن ویژه ای مانند CKEditor یا TinyMCE هم به جاوااسکریپت نوشته شده اند که چیزی شبیه Word را روی صفحه ی وب نمایش می دهند و خروجی آنها کدهای اچ تی ام هستند. اما بعد از خواندن و نمایش متن آرایش شده چیزی که روی صفحه خواهید دید، ممکن است شبیه این باشد:

سلام
 خط دوم

یعنی به جای اِعمال شدن تگ ها، درست عین متن روی صفحه دیده می شوند. برای خنثا کردن این حالت کافیست بخش نمایش محتوا را به صورت زیر تغییر دهید:

```
@* تبدیل رشته به اچ تی ام ال *@      @Html.Raw(content)
```

اما همانطور که به ذهن شما هم رسیده، به طور معمول امکان ویرایش نباید بلافاصله بعد از اجرای برنامه در دسترس کاربران باشد. یعنی با ورود به هیچ سایتی، تا قبل از Login یا همان احراز هویت کسی قادر به ایجاد تغییرات نیست.

شناسایی کاربر

ایده ی چنین چیزی چطور محقق خواهد شد؟ شاید اگر هدف تنها نمایش یک پیام ساده در صفحه باشد، قرار دادن یک تکست باکس، و کنترل مقدار ورودی در آن به عنوان رمز عبور، با رفرش شدن صفحه بعد از ارسال این رمز، کار ساده ای باشد. با یک دستور if می شود فهمید که کاربر رمز ثابت و تعریف شده ی ما را در برنامه درست وارد کرده یا نه و در این صورت بخش هایی از صفحه را به او نشان بدهیم. اما اگر این کنترل را در صفحات دیگر بخواهیم چی می شود؟ آیا منطقی است که برای مشاهده ی هر صفحه یا مثلاً بخش های ورود یا ویرایش اطلاعات آن صفحات، کاربر را مجبور به لاگین یا وارد کردن مجدد رمز کنیم!؟

با همین چالش ساده، زبانهای طراحی شده برای وب به مفهومی به نام "جلسه" نیاز پیدا کرده اند. به این معنی که هر کاربری هنگام ورود به یک وب اپلیکیشن بتواند حافظه ای در سِرور را به عنوان فضای نگهداری این ملاقات به خود اختصاص دهد. به این ترتیب **در این جلسه ی اختصاصی که نامش در کدنویسی سمت وب Session است، می توان متغیرهایی را مستقل از اینکه در چه صفحه ای حضور داریم تعریف کرد! حالا از تمام صفحات به این متغیر که برای آن کاربر تعریف شده دسترسی داریم.** یعنی می توان در صورتی که مثلاً کاربر عمل Login را درست انجام داده باشد متغیری به همین نام را برای مقدار True قرار داد و فقط در صفحات آن را چک کرد و دسترسی های مورد نیاز را فراهم کرد.

بخش Session در دات نت به طور پیشفرض غیر فعال است. دلیل این امر بهینه سازی هایی است که برای کاهش حجم کتابخانه ها و بالابردن سرعت دات نت در نظر گرفته شده است. همچنین راهکارهای جدیدتری نیز برای ایجاد دسترسی کاربران وجود دارد که می توان بدون سشن ایجاد کرد. به همین دلیل استفاده از آن را انتخابی و قابل تنظیم کرده اند. حالا وقت آن است که به سراغ یکی از فایل های غریبه ی پروژه ی تحت وب خود به نام Startup.cs برویم:

```
public void ConfigureServices(IServiceCollection services)
    {
        services.AddSession();
        services.AddRazorPages();
    }
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseRouting();
        app.UseSession();
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
    }
```

البته که در اینجا تنظیمات زیادی در مورد پروژه ی تحت وب ما به چشم می خورد. اما ما دو بخش مشخص شده را به آن اضافه می کنیم. یعنی یک بار در بخش ConfigureServices این سرویس یا خدمت را با service.Add اضافه کرده و بار دوم آن را در بخش Configure با app.Use مورد بهره برداری در پروژه قرار می دهیم. البته اد کردن یک سرویس را می توان با تنظیمات خاص ان سرویس هم انجام داد. یعنی مثلاً اینکه سشن های ما طی چه زمانی منقضی شده و با دست نزدن کاربر به سیستم و رفرش نشدن صفحات بعد از ۳۰ دقیقه به طور خودکار این متغیرها پاک شوند:

```
services.AddSession(options =>
      {
        options.IdleTimeout = TimeSpan.FromMinutes(30);
       });
```

بعد از تغییراتی که در فایل استارت آپ پروژه برای فعال کردن قابلیت سشن انجام دادیم، لازم است کتابخانه ی کدی را هم که دستورات مربوط به کار با سشن ها در آن وجود دارد به صفحه و مدل مورد نظرمان اضافه نماییم. پس در دومین حرکت مثلاً برای Index.cshtml در بالای صفحه کنار using های دیگر خواهیم داشت:

```
@using Microsoft.AspNetCore.Http
```

در فایل مدل مرتبط با صفحه یعنی Index.cshtml.cs هم داریم:

```
using Microsoft.AspNetCore.Http;
```

این کتابخانه بر خلاف کتابخانه هایی که تا این لحظه شناخته ایم و با System. شروع می شدند، با Microsoft.AspNetCore. شروع شده است. این نامگذاری به خوبی نشان می دهد که کدهای نوشته شده، مخصوص وب و ASP هستند و در اینجا کلاس Http که آن را use کرده ایم در هیچ پروژه از نوع دیگری مثل Console قابل بهره برداری نیست.

حالا آماده ایم که نمونه هایی از متغیرهای فراگیر تعریف شده در سشن را به طور ساده تست کنیم. روش ساختن یا سِت کردن مقدار برای متغیرهای username و userId را به طور مثال به صورت زیر در فایل Index.cshtml.cs به صورت زیر انجام دهید:

```
public void OnGet()
     {
     HttpContext.Session.SetString ("username", "abcd");
     HttpContext.Session.SetInt32 ("userId", 1);
    }
```

به همین راحتی، دو متغیر از نوع رشته ای و عددی را تعریف و مقدار دهی کرده ایم که نه تنهای در صفحه ی Index بلکه در صفحات دیگر نیز قابل دسترس هستند! برای تست به سراغ صفحه ی Privacy.cshtml بروید. فراموش نکنید که در ابتدای آن کتابخانه ی کار با سشن ها را یوز کرده باشید:

```
@using Microsoft.AspNetCore.Http
```

حالا کافیست در این صفحه این بار به جای Set از Get برای فراخوانی یک متغیر از نوع سشن بهره گرفته و مثلاً آن را نمایش داده یا به متغیری نسبت بدهید:

```
@{
string logedUser = HttpContext.Session.GetString ("username");
}
@logedUser
```

به محض اجرای پروژه و رفتن به صفحه ی Privacy خواهید دید که مقدار "abcd" که در صفحه ی ایندکس برای username ست شده، در اینجا دریافت و نمایش داده شده.

به همین راحتی می توان فرمی را برای ورود کاربر، دریافت رمز، چک کردن رمز در صورت برابر بودنش با مورد دلخواه ما، و نمایش بخش های ویرایش در صورت صحت رمز ساخت. فقط در هنگام چک کردن بهتر است که ابتدا مقادیر یک متغیر سشن را به یک متغیر معمولی در صفحه نسبت دهید تا کد خواناتری داشته باشید:

```
@{
    string user = HttpContext.Session.GetString("username");
    if (user == "abcd")
    {
        HttpContext.Session.SetInt32("userId", 2);
    }
}
```

در مثال بالا، اگر مقدار username برابر abcd باشد (که هست) مقدار userId به ۲ تغییر داده می شود. می توان هر عملیات دیگری را درون این شرط قرار داده یا حتا شرط های دیگری نوشت که بر اساس آن اعتبارسنجی های مد نظر ما در صفحات صورت پذیرد.

سخن پایانی

هر چند به شکل روتین موارد بسیاری برای یادگیری باقی مانده است. مثلاً استفاده از انواع دیتابیس مانند SQLite در پروژه های تجاری مطرح است و به صورت پکیج های اضافه به پروژه ی سی شارپ شما ملحق می شوند. همچنین دستورهای کاربردی بسیاری در سی شارپ وجود دارند که به منظور تولید انواع امکانات مانند آپلود فایل در پروژه ها به کار می روند. اما آنچه که در این مجموعه مطرح شد برای تکمیل مطالعات و شناخت امکانات بیشتر در آینده کفایت می کند. شما در این لحظه بعد از فهم مطالب مطرح شده می توانید خود را برنامه نویس سی شارپ بدانید. همچنین می توانید برای ساخت برنامه های حرفه ای تر در اینترنت کدهای مختلفی را جستجو کرده و برای فهم و به کار بردن آنها تلاش کرده یا با همین مطالب دست به ساختن برنامه های محاسباتی و اطلاعاتی بزنید. همچنین قادر به درک کدهای نوشته شده توسط دیگران بوده و می توانید در تکمیل یا تغییر پروژه های آماده شرکت کنید.