iscte

INSTITUTO
UNIVERSITÁRIO
DE LISBOA

emprego
digital

UPskill

Contents

What will this module be about?

- Introduction to .NET Core
- Setting up our working environment
- First .NET Core application
- .NET Core CLI
- Introduction to C# Language

Introduction to .NET Core

.NET Core

- .NET Core is an open-source, general-purpose development platform
    - Uses MIT and Apache 2 licenses

- Create apps for Windows, macOS, and Linux (cross platform)
    - for x64, x86, ARM32, and ARM64 processors
    - using multiple programming languages
    - Consistent across environments

.NET Core

- Frameworks and APIs are provided for cloud, IoT, client UI, and machine learning:
  - Cloud apps with ASP.NET Core
  - Mobile apps with Xamarin
  - IoT apps with System.Device.GPIO
  - Windows client apps with WPF and Windows Forms
  - Machine learning ML.NET

.NET Core

- Implements .NET Standard
  - formal specification of .NET APIs that are intended to be available on all .NET implementations

- Created with the goal to componentize .NET Core
  - to support modern technologies and
  - to have fewer dependencies - deployment requires only those packages that your application needs

.NET Core is composed of:

- The .NET Core runtime
  - provides a type system, assembly loading, a garbage collector, native interop, and other services.
  - .NET Core framework libraries provide primitive data types, app composition types, and fundamental utilities.

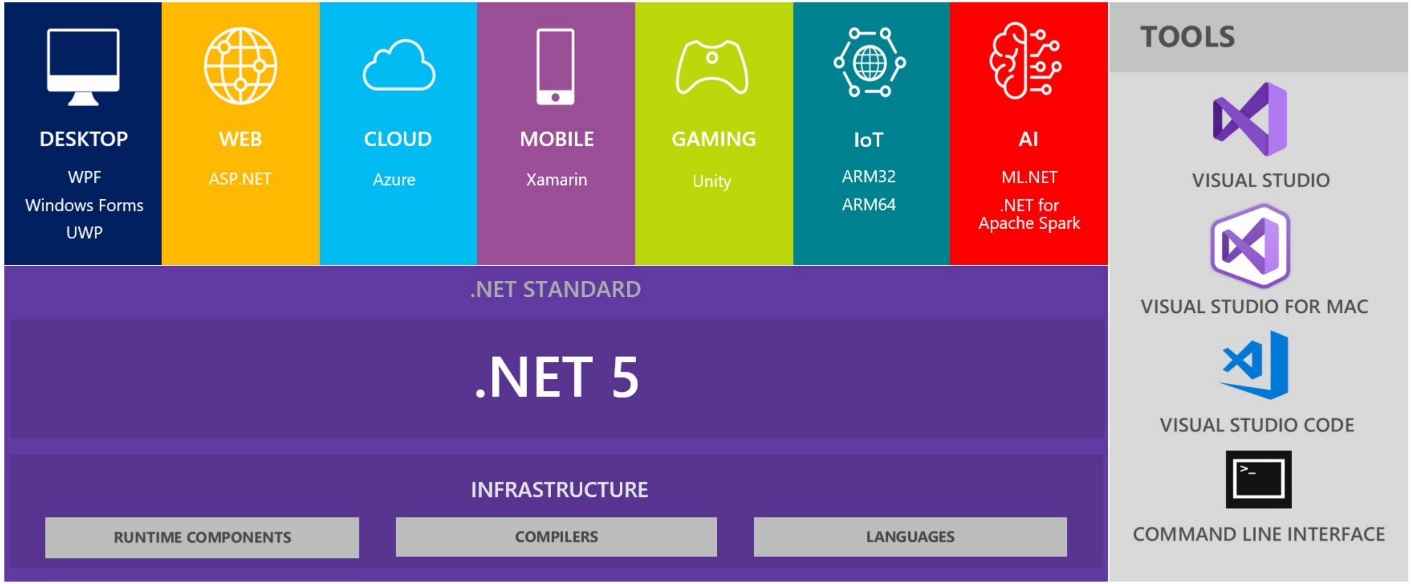
- The ASP.NET Core runtime
  - provides a framework for building modern, cloud-based, internet-connected apps, such as web apps, IoT apps, and mobile backends.

.NET Core is composed of:

- The **.NET Core SDK** and **language compilers** (Roslyn and F#) that enable the .NET Core developer experience.

- The **dotnet** command
  - used to launch .NET Core apps and CLI commands.

.NET – A unified platform



DESKTOP

WPF

Windows Forms

UWP

WEB

ASP.NET

CLOUD

Azure

MOBILE

Xamarin

GAMING

Unity

IoT

ARM32

ARM64

AI

ML.NET

.NET for Apache Spark

.NET STANDARD

.NET 5

INFRASTRUCTURE

RUNTIME COMPONENTS

COMPILERS

LANGUAGES

TOOLS

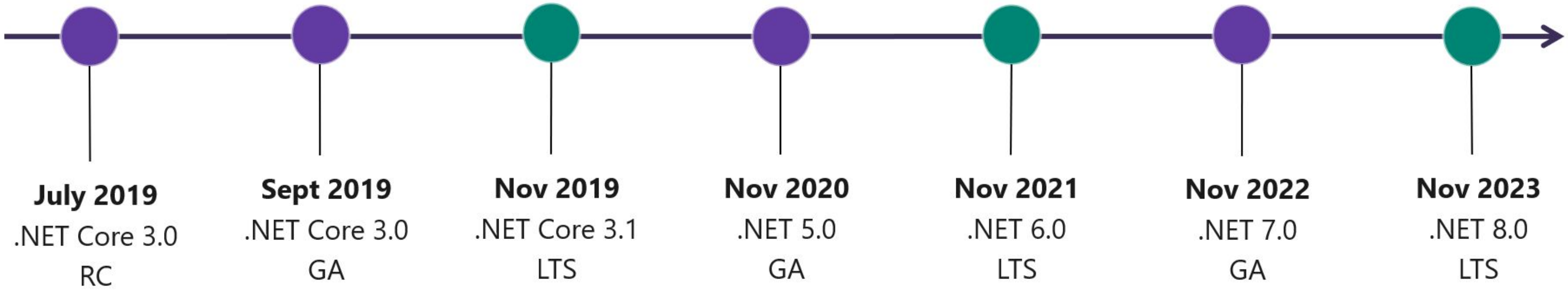
VISUAL STUDIO

VISUAL STUDIO FOR MAC

VISUAL STUDIO CODE

COMMAND LINE INTERFACE

.NET Schedule



July 2019
.NET Core 3.0
RC

Sept 2019
.NET Core 3.0
GA

Nov 2019
.NET Core 3.1
LTS

Nov 2020
.NET 5.0
GA

Nov 2021
.NET 6.0
LTS

Nov 2022
.NET 7.0
GA

Nov 2023
.NET 8.0
LTS

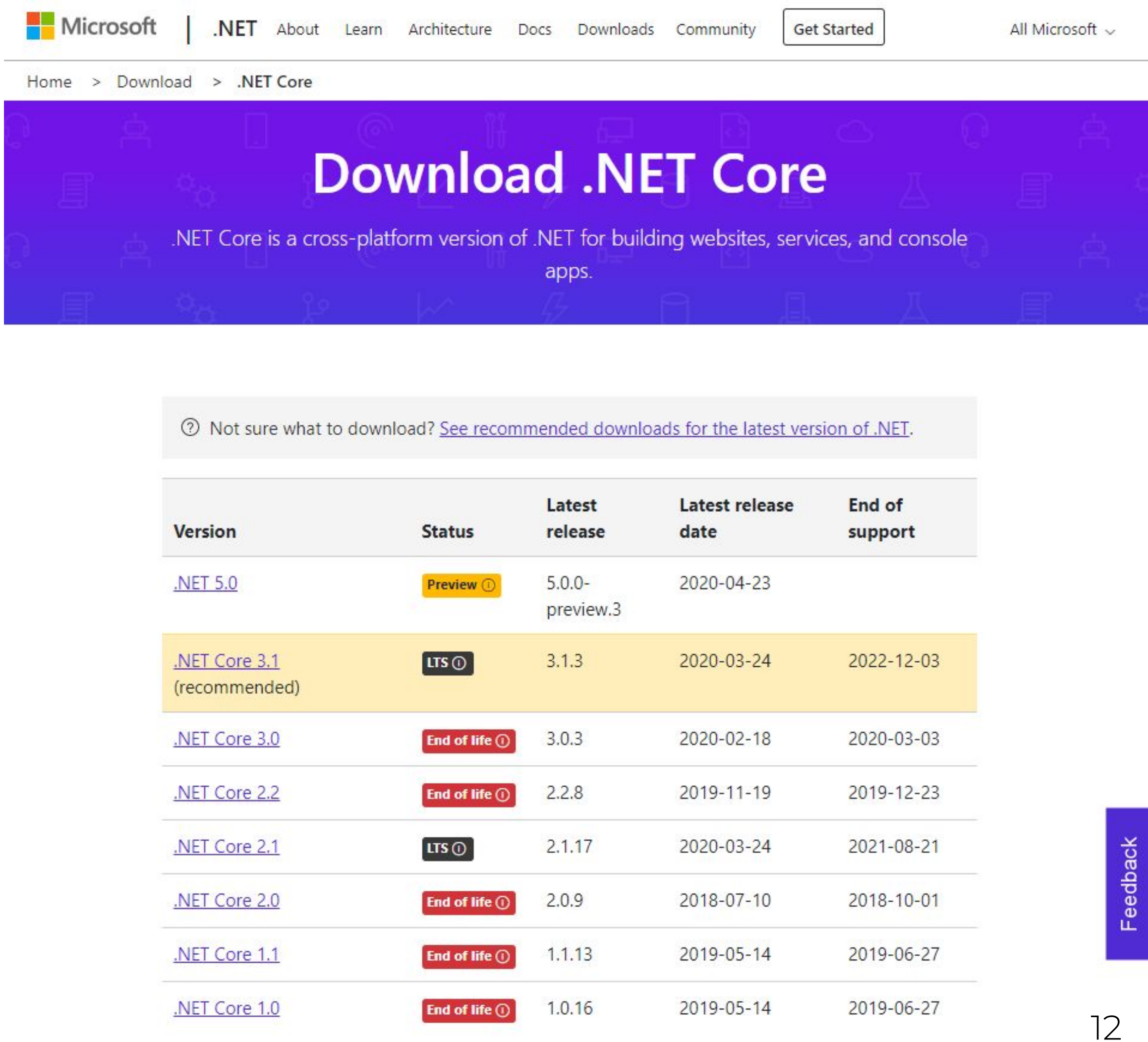
- .NET Core 3.0 release in September
- .NET Core 3.1 = Long Term Support (LTS)
- .NET 5.0 release in November 2020
- Major releases every year, LTS for even numbered releases
- Predictable schedule, minor releases if needed

Introducing .NET 5 | .NET Blog

Setting up our working environment

Tools

- .NET Core Runtime or SDK?
    - [Download .NET (Linux, macOS, and Windows)]()

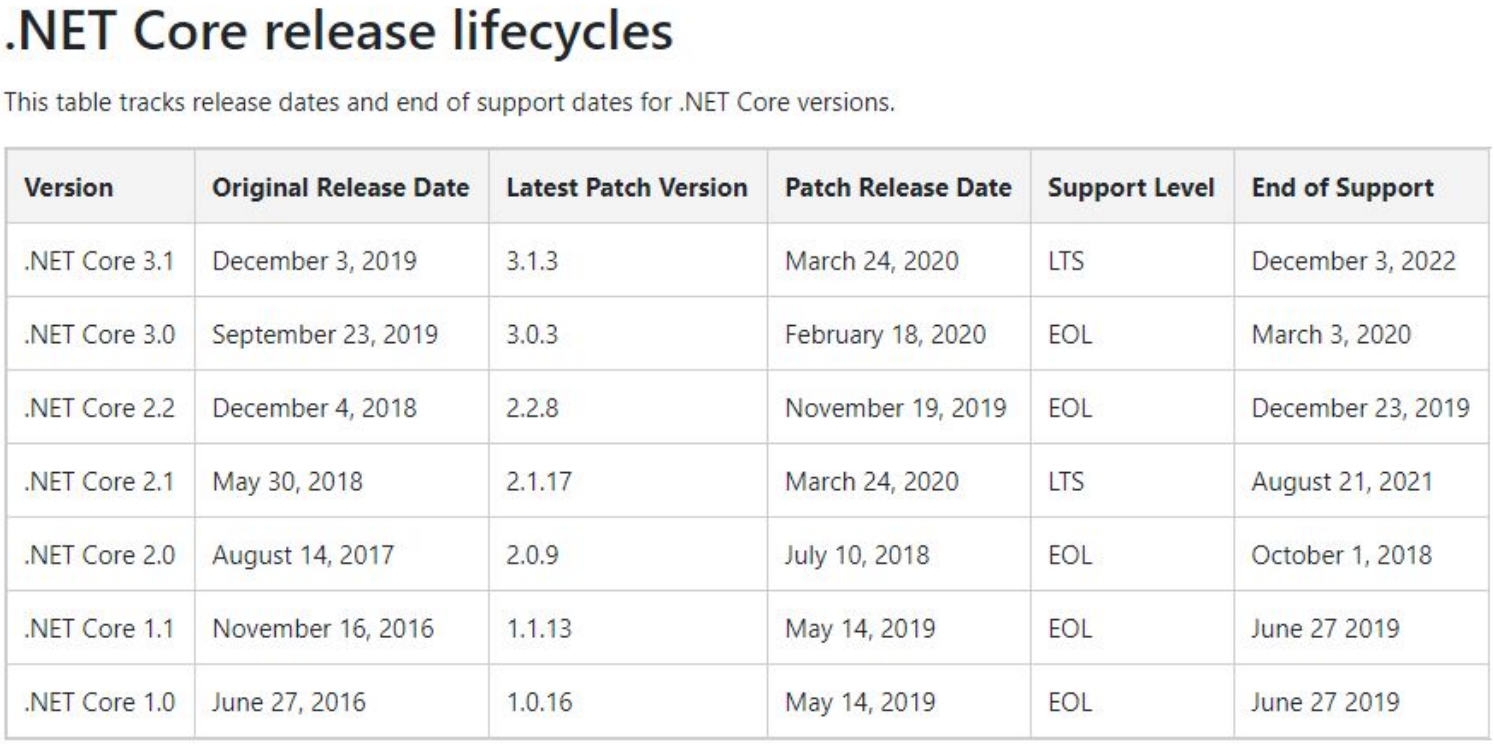


Download .NET Core

.NET Core is a cross-platform version of .NET for building websites, services, and console apps.

ⓘ Not sure what to download? See recommended downloads for the latest version of .NET.

Version	Status	Latest release	Latest release date	End of support
.NET 5.0	Preview ⓘ	5.0.0-preview.3	2020-04-23	
.NET Core 3.1 (recommended)	LTS ⓘ	3.1.3	2020-03-24	2022-12-03
.NET Core 3.0	End of life ⓘ	3.0.3	2020-02-18	2020-03-03
.NET Core 2.2	End of life ⓘ	2.2.8	2019-11-19	2019-12-23
.NET Core 2.1	LTS ⓘ	2.1.17	2020-03-24	2021-08-21
.NET Core 2.0	End of life ⓘ	2.0.9	2018-07-10	2018-10-01
.NET Core 1.1	End of life ⓘ	1.1.13	2019-05-14	2019-06-27
.NET Core 1.0	End of life ⓘ	1.0.16	2019-05-14	2019-06-27



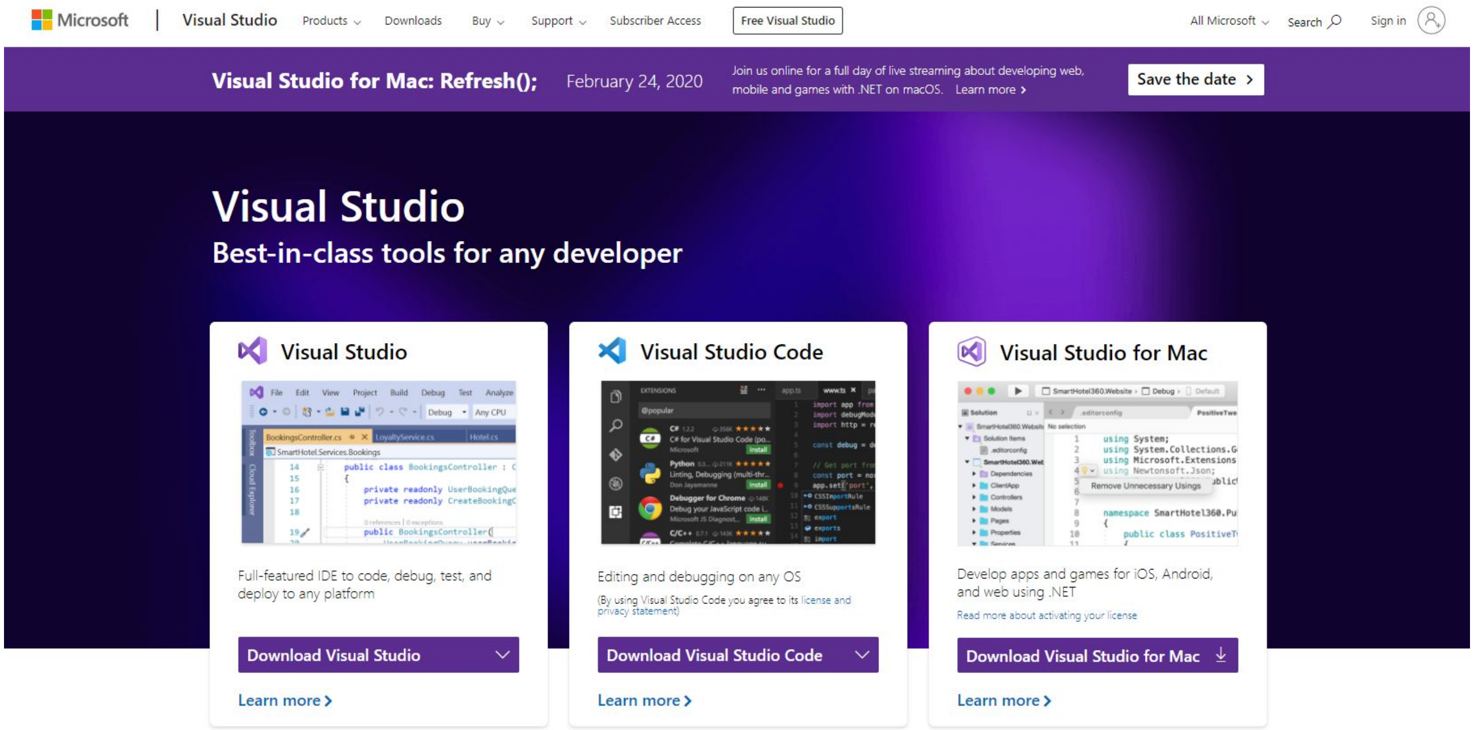
.NET Core release lifecycles

This table tracks release dates and end of support dates for .NET Core versions.

Version	Original Release Date	Latest Patch Version	Patch Release Date	Support Level	End of Support
.NET Core 3.1	December 3, 2019	3.1.3	March 24, 2020	LTS	December 3, 2022
.NET Core 3.0	September 23, 2019	3.0.3	February 18, 2020	EOL	March 3, 2020
.NET Core 2.2	December 4, 2018	2.2.8	November 19, 2019	EOL	December 23, 2019
.NET Core 2.1	May 30, 2018	2.1.17	March 24, 2020	LTS	August 21, 2021
.NET Core 2.0	August 14, 2017	2.0.9	July 10, 2018	EOL	October 1, 2018
.NET Core 1.1	November 16, 2016	1.1.13	May 14, 2019	EOL	June 27 2019
.NET Core 1.0	June 27, 2016	1.0.16	May 14, 2019	EOL	June 27 2019

https://dotnet.microsoft.com/platform/support/policy/dotnet-core

Integrated Development Environment (IDE)



Microsoft | Visual Studio Products ⌄ Downloads Buy ⌄ Support ⌄ Subscriber Access Free Visual Studio All Microsoft ⌄ Search Sign in

Visual Studio for Mac: Refresh(); February 24, 2020 Join us online for a full day of live streaming about developing web, mobile and games with .NET on macOS. Learn more › Save the date ›

Visual Studio

Best-in-class tools for any developer

Visual Studio

Full-featured IDE to code, debug, test, and deploy to any platform

Download Visual Studio ⌄

Learn more ›

Visual Studio Code

Editing and debugging on any OS

(By using Visual Studio Code you agree to its license and privacy statement)

Download Visual Studio Code ⌄

Learn more ›

Visual Studio for Mac

Develop apps and games for iOS, Android, and web using .NET

Read more about activating your license

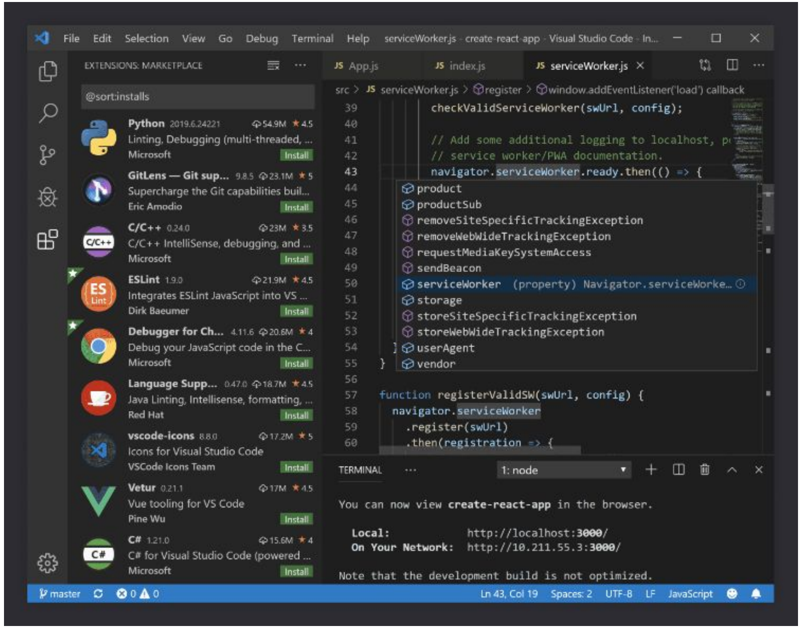
Download Visual Studio for Mac ⤓

Learn more ›

Our choice - Microsoft Visual Studio Code

- Modern and lightweight code editor
- Runs on all common operating systems, including Windows, macOS , and Linux
- Extensive and growing set of extensions

https://code.visualstudio.com/


File
Edit
Selection
View
Go
Debug
Terminal
Help
serviceWorker.js - create-react-app - Visual Studio Code - In...
EXTENSIONS: MARKETPLACE
@sort:installs
JS App.js
JS index.js
JS serviceWorker.js
src > JS serviceWorker.js > register > window.addEventListener('load') callback
product
productSub
removeSiteSpecificTrackingException
removeWebWideTrackingException
requestMediaKeySystemAccess
sendBeacon
serviceWorker (property) Navigator.serviceWorke...
storage
storeSiteSpecificTrackingException
storeWebWideTrackingException
userAgent
vendor
TERMINAL
1: node
master
0 0
Ln 43, Col 19
Spaces: 2
UTF-8
LF
JavaScript

Python 2019.6.24221 54.9M ★4.5
Linting, Debugging (multi-threaded, ...
Microsoft Install

GitLens — Git sup... 9.8.5 23.1M ★5
Supercharge the Git capabilities buil...
Eric Amodio Install

C/C++ 0.24.0 23M ★3.5
C/C++ IntelliSense, debugging, and ...
Microsoft Install

ESLint 1.9.0 21.9M ★4.5
Integrates ESLint JavaScript into VS ...
Dirk Baeumer Install

Debugger for Ch... 4.11.6 20.6M ★4
Debug your JavaScript code in the C...
Microsoft Install

Language Supp... 0.47.0 18.7M ★4.5
Java Linting, Intellisense, formatting, ...
Red Hat Install

vscode-icons 8.8.0 17.2M ★5
Icons for Visual Studio Code
VSCode Icons Team Install

Vetur 0.21.1 17M ★4.5
Vue tooling for VS Code
Pine Wu Install

C# 1.21.0 15.6M ★4
C# for Visual Studio Code (powered ...
Microsoft Install

```
      checkValidServiceWorker(swUrl, config);

      // Add some additional logging to localhost, p
      // service worker/PWA documentation.
      navigator.serviceWorker.ready.then(() => {
    }
  }

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
```

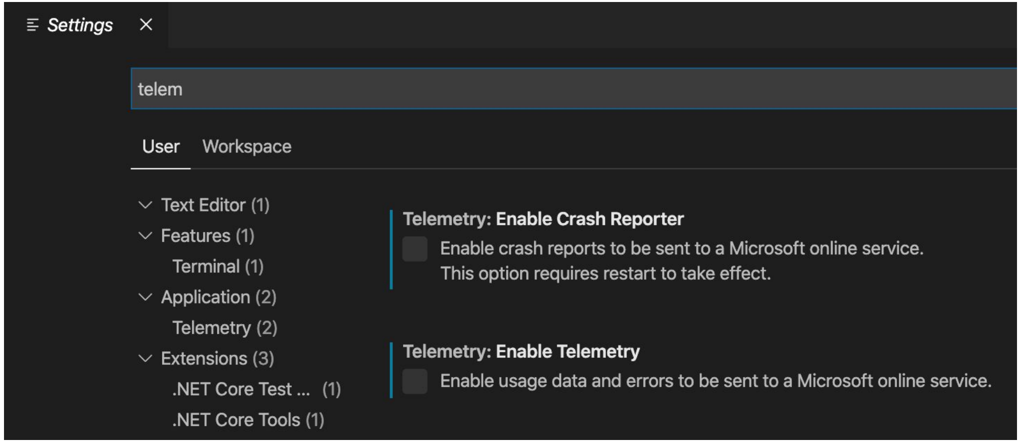
```
You can now view create-react-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://10.211.55.3:3000/

Note that the development build is not optimized.
```

Our choice - Microsoft Visual Studio Code

- Some useful Extensions:
  - C# (Microsoft - Powered by Omnisharp)
  - .NET Core Tools
  - .NET Core Starter's Pack
  - Visual Studio Intellicode
  - Nuget Package Manager
  - SQLite
  - (optionally) Vetur
  - (optionally) GraphQL for VSCode
- Keyboard shortcuts for Windows

Visual Studio Code - First time

- Disable Telemetry
  - Settings > search "telem"
  - Enable Crash Reporter
  - Enable Telemetry



Settings ✕

telem

User Workspace

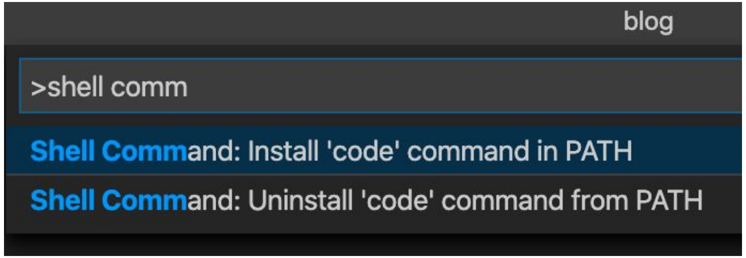
Telemetry: Enable Crash Reporter

□ Enable crash reports to be sent to a Microsoft online service. This option requires restart to take effect.

Telemetry: Enable Telemetry

□ Enable usage data and errors to be sent to a Microsoft online service.

- *MacOSX*
  Add code to terminal
  - Command+shift+p
  - Search "shell comm"



>shell comm

Shell Command: Install 'code' command in PATH

Shell Command: Uninstall 'code' command from PATH

First .NET Core Application

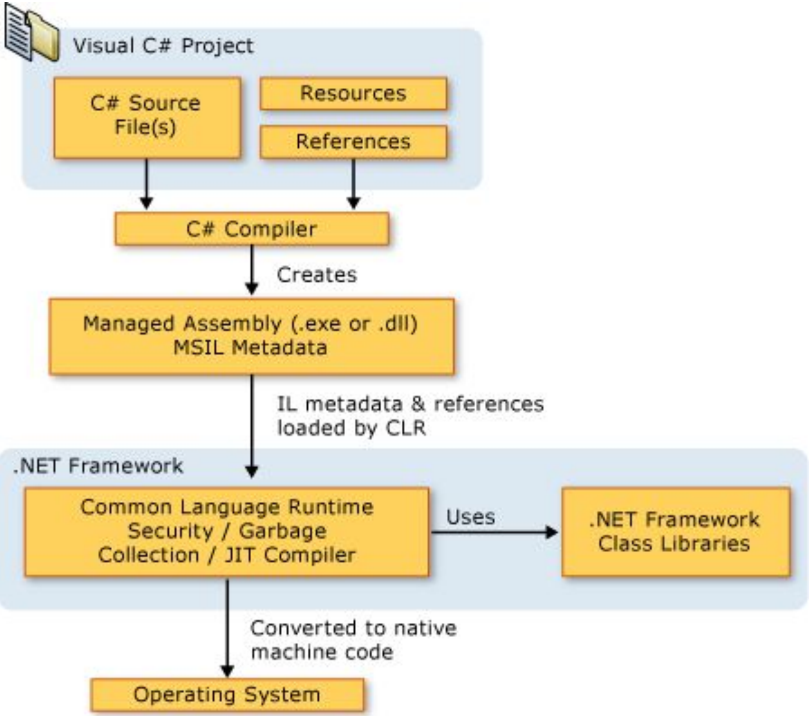
First Application

- In terminal, type:
    a. `dotnet new console`
    b. `dotnet run`

- Explore the project by opening it in VSCode

- Can you print your name after “Hello“?

.NET Framework Platform Architecture


Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

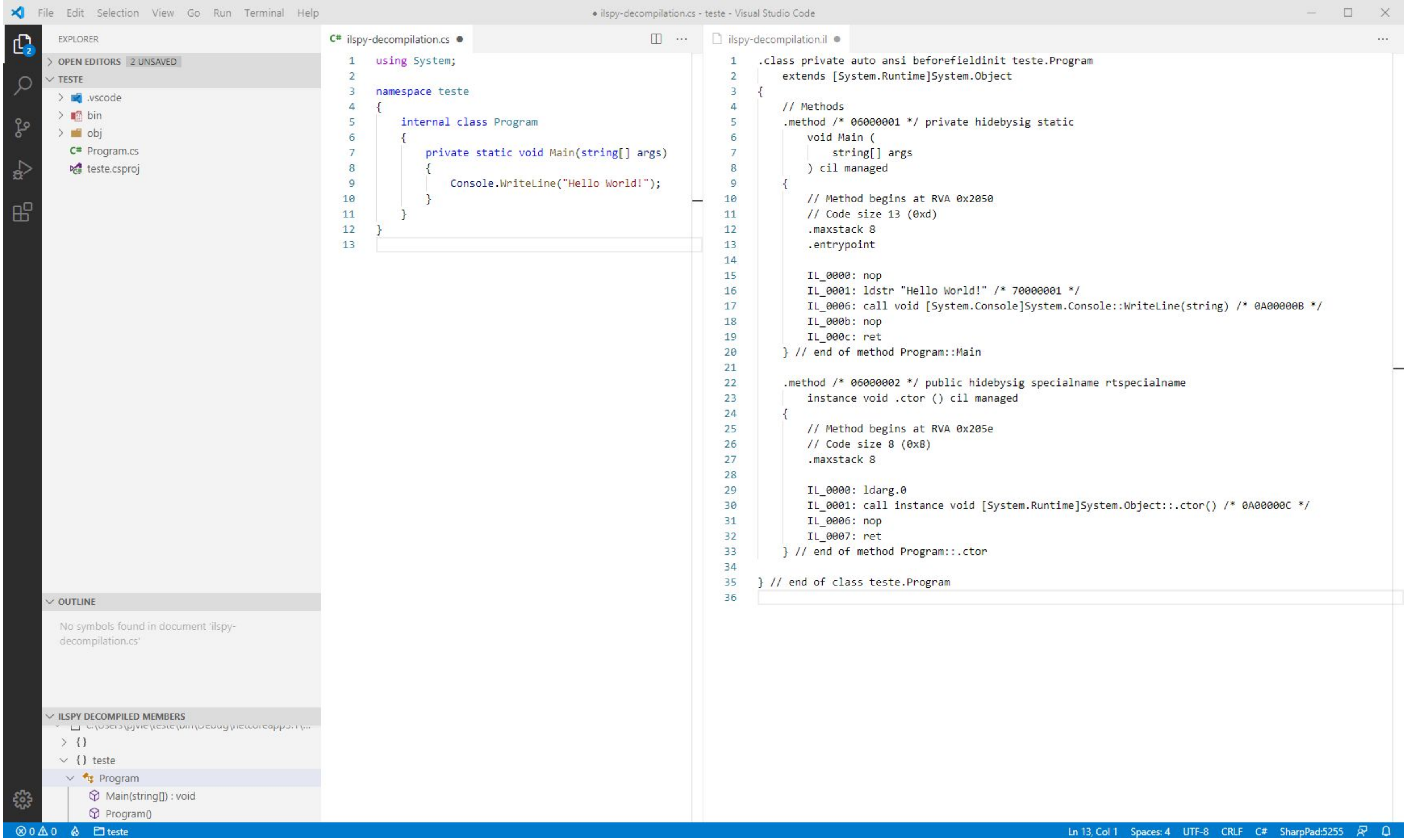
How does it work? Compile-time

- C# compiler (Roslyn) converts source code into intermediate language (IL)

- Stores the IL in an assembly (a DLL or EXE file)

How does it work? Runtime

- Executed by .NET Core’s virtual machine, known as CoreCLR
    1. loads the IL code from the assembly,
    2. the just-in-time (JIT) compiler compiles it into native CPU instructions, and
    3. then it is executed by the CPU on your machine



No symbols found in document 'ilspy-decompilation.cs'

```csharp
using System;

namespace teste
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}

```

```
.class private auto ansi beforefieldinit teste.Program
    extends [System.Runtime]System.Object
{
    // Methods
    .method /* 06000001 */ private hidebysig static
        void Main (
            string[] args
        ) cil managed
    {
        // Method begins at RVA 0x2050
        // Code size 13 (0xd)
        .maxstack 8
        .entrypoint

        IL_0000: nop
        IL_0001: ldstr "Hello World!" /* 70000001 */
        IL_0006: call void [System.Console]System.Console::WriteLine(string) /* 0A00000B */
        IL_000b: nop
        IL_000c: ret
    } // end of method Program::Main

    .method /* 06000002 */ public hidebysig specialname rtspecialname
        instance void .ctor () cil managed
    {
        // Method begins at RVA 0x205e
        // Code size 8 (0x8)
        .maxstack 8

        IL_0000: ldarg.0
        IL_0001: call instance void [System.Runtime]System.Object::.ctor() /* 0A00000C */
        IL_0006: nop
        IL_0007: ret
    } // end of method Program::.ctor

} // end of class teste.Program

```

Hands On

15-min tutorial

- Get started with C# and Visual Studio Code - .NET Core

Quiz

1. Why can a programmer use different languages, for example, C# and F#, to write applications that run on .NET Core?
2. What do you type at the prompt to create a console app?
3. What do you type at the prompt to build and execute C# source code?
4. Is .NET Core better than .NET Framework?
5. Where would you look for help about a C# keyword?

.NET Core CLI

.NET Core CLI

- The .NET Core command-line interface (CLI) is a cross-platform toolchain for developing, building, running, and publishing .NET Core applications

- > dotnet -h

.NET Core CLI

- CLI command structure consists of:
  - the driver ("dotnet") - two responsibilities: either running a framework-dependent app or executing a command

  - the command - performs an action. For example, dotnet build builds code. dotnet publish publishes code.

  - possibly command arguments and options.

.NET Core CLI

- **`dotnet new`** - Creates a new project, configuration file, or solution based on the specified template.
- **`dotnet restore`** - Restores the dependencies and tools of a project.
- **`dotnet build`** - Builds a project and all of its dependencies.
- **`dotnet clean`** - Cleans the output of a project.
- **`dotnet run`** - Runs source code without any explicit compile or launch commands.

Hands On

- O que faz o comando `dotnet new --list` ?

- E este ?

```
dotnet new --install Microsoft.AspNetCore.SpaTemplates::*
```

  - https://dotnetnew.azurewebsites.net/

Introduction to C# language

Introduction to the C# language

- C# is an elegant and type-safe object-oriented language
  - enables developers to build a variety of secure and robust applications that run on the .NET Framework.

- Can be used to create:
  - Windows client applications, XML Web services, distributed components, client-server applications, database applications.

https://docs.microsoft.com/en-us/dotnet/csharp/

Introduction to the C# language

- C# has its roots in the C family of languages

- C# is standardized by ECMA International as the ECMA-334 standard and by ISO/IEC as the ISO/IEC 23270 standard.

- Microsoft's C# compiler for the .NET Framework is a conforming implementation of both of these standards.

History of C#

- C# 1.0 was released in 2002
- Current version C# 8.0 was released in 2019

Program Structure

- Key organizational concepts in C# are:
    - programs,
    - namespaces,
    - types,
    - members, and
    - Assemblies

- C# programs consist of one or more source files

Program Structure

- Programs declare types, which contain members and can be organized into namespaces.
    - Classes and interfaces are examples of types
    - Fields, methods, properties, and events are examples of members

- When C# programs are compiled, they're physically packaged into assemblies.
    - Assemblies typically have the file extension .exe or .dll, depending on whether they implement applications or libraries, respectively.

Types and variables

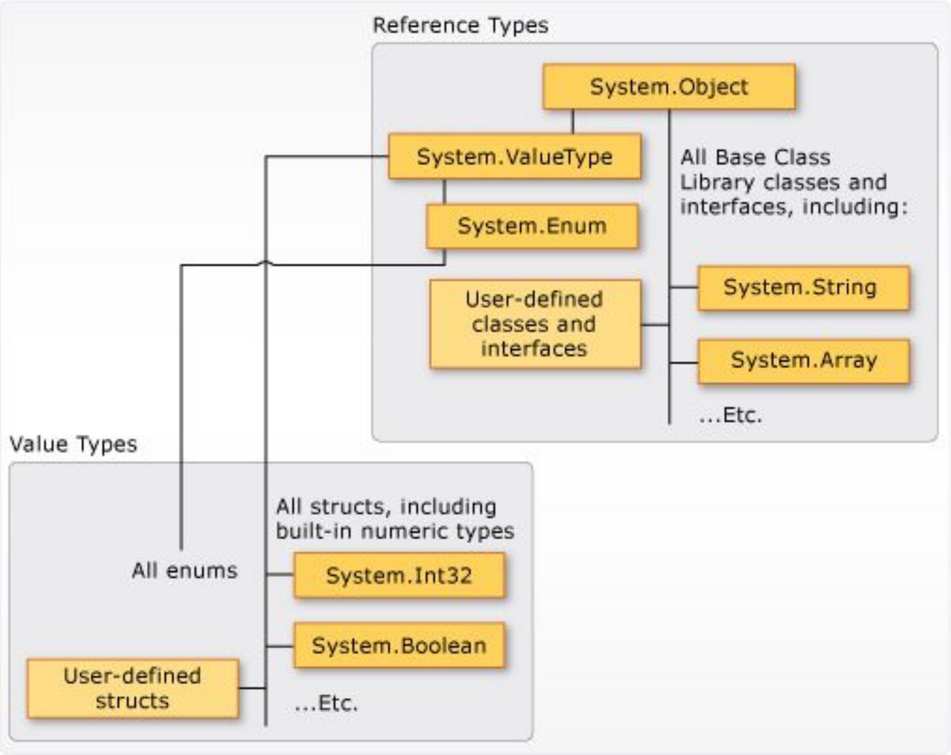
- There are two kinds of types in C#: **value types** and **reference types**.
    - Value types directly contain their data
    - Reference types store references to their data, the latter being known as objects.

Types and variables

- C#'s value types are further divided into:
  - simple types,
  - enum types,
  - struct types, and
  - nullable value types

- C#'s reference types are further divided into:
  - class types,
  - interface types,
  - array types, and
  - delegate types.

Types and variables



Why C#?

- C# is a member of the C family of programming languages (e.g., C, Objective C, C++, Java) and, therefore, share a similar syntax

- C# supports a number of features traditionally found in various functional languages (e.g., LISP or Haskell) such as **lambda expressions** and **anonymous types**

- Supports Language Integrated Query (LINQ)

Why C#?

- No pointers required! C# programs typically have no need for direct pointer manipulation
- Automatic memory management through garbage collection
- Formal syntactic constructs for classes, interfaces, structures, enumerations, and delegates
- The C++-like ability to overload operators for a custom type
- Support for attribute-based programming

Why C#?

- .NET 2.0 (2005)
  - Build generic types and generic members
  - Support for anonymous methods
  - define a single type across multiple code files

- .NET 3.5 (2008)
  - Support for strongly typed queries (e.g., LINQ)
  - Inclusion of a lambda operator (=>)
  - A new object initialization syntax

Why C#?

- .NET 4.0 (2010)
  - Support for optional method parameters
  - Support for dynamic lookup of members at runtime via the dynamic keyword

- .NET 4.6 (2015)
  - Single-line method implementations using the C# lambda operator
  - A null conditional operator, which helps check for null parameters in a method implementation
  - A new string formatting syntax termed string interpolation
  - The ability to filter exceptions using the new when keyword

Managed vs. Unmanaged Code

- Managed Code
  - Code targeted at .NET runtime
  - The binary unit is called *assembly*

- Unmanaged Code
  - Code that cannot be directly hosted by .NET runtime

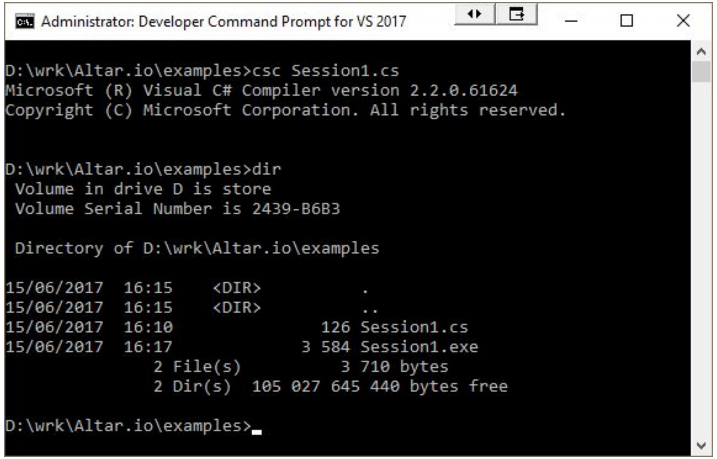
- .NET binaries contain platform-agnostic Intermediate Language (IL or CIL) and type metadata

Code Labs *up and running*

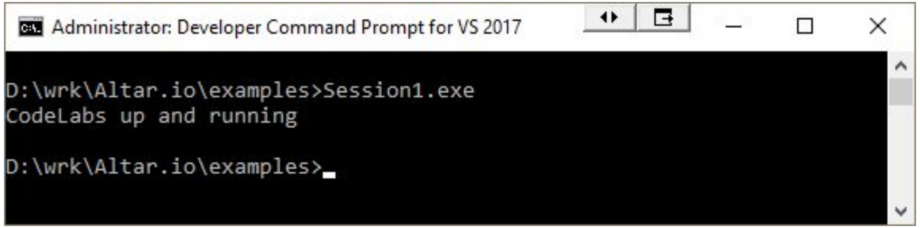
```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Code Labs
up and running");
    }
}
```

Session1.cs

csc Session1.cs



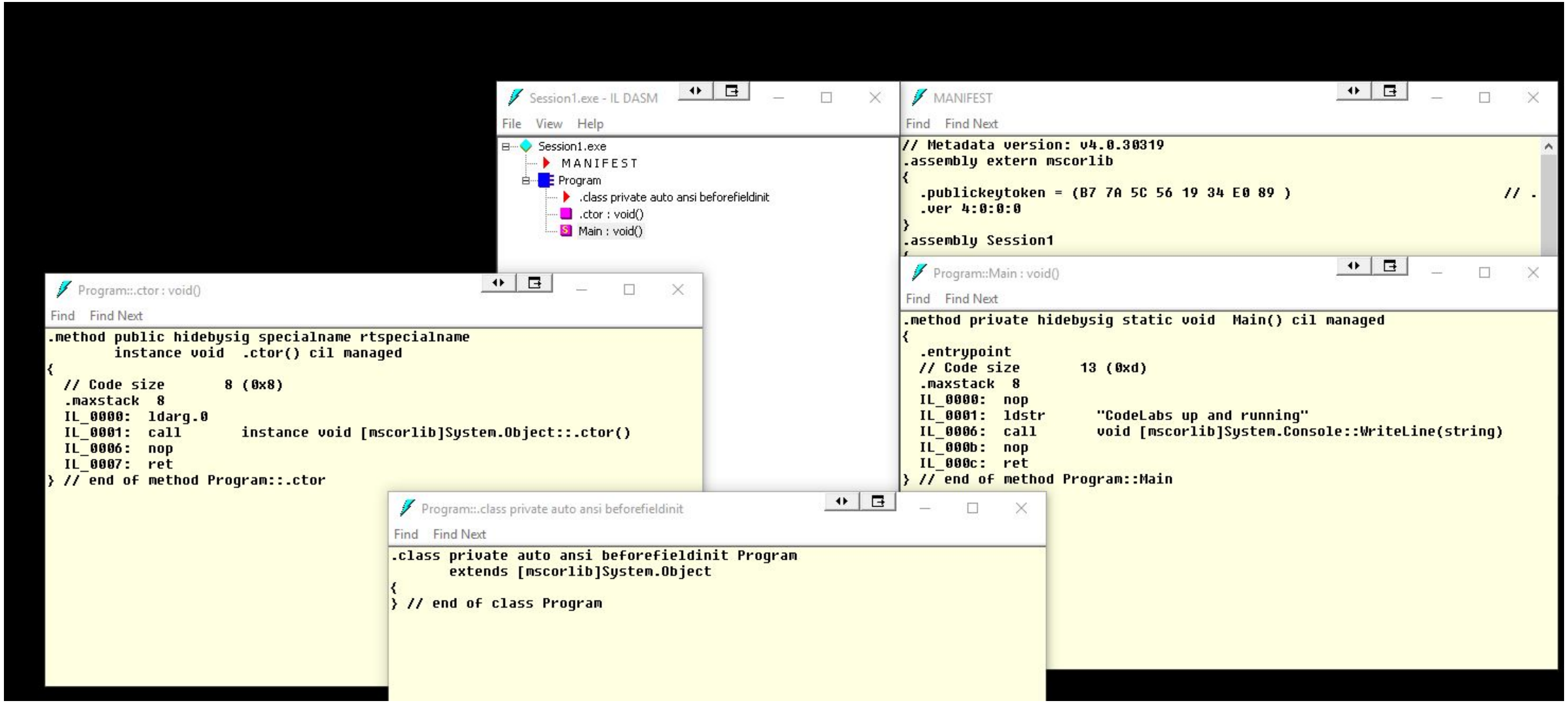
```
D:\wrk\Altar.io\examples>csc Session1.cs
Microsoft (R) Visual C# Compiler version 2.2.0.61624
Copyright (C) Microsoft Corporation. All rights reserved.


D:\wrk\Altar.io\examples>dir
 Volume in drive D is store
 Volume Serial Number is 2439-B6B3

 Directory of D:\wrk\Altar.io\examples

15/06/2017  16:15    <DIR>          .
15/06/2017  16:15    <DIR>          ..
15/06/2017  16:10               126 Session1.cs
15/06/2017  16:17             3 584 Session1.exe
               2 File(s)          3 710 bytes
               2 Dir(s)  105 027 645 440 bytes free

D:\wrk\Altar.io\examples>_
```



```
D:\wrk\Altar.io\examples>Session1.exe
CodeLabs up and running

D:\wrk\Altar.io\examples>_
```

Code Labs *up and running*

Ildasm
Session1.exe



```
// Metadata version: v4.0.30319
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .
  .ver 4:0:0:0
}
.assembly Session1
```

```
.method private hidebysig static void  Main() cil managed
{
  .entrypoint
  // Code size       13 (0xd)
  .maxstack  8
  IL_0000:  nop
  IL_0001:  ldstr      "CodeLabs up and running"
  IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000b:  nop
  IL_000c:  ret
} // end of method Program::Main
```

```
.method public hidebysig specialname rtspecialname
        instance void  .ctor() cil managed
{
  // Code size       8 (0x8)
  .maxstack  8
  IL_0000:  ldarg.0
  IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
  IL_0006:  nop
  IL_0007:  ret
} // end of method Program::.ctor
```

```
.class private auto ansi beforefieldinit Program
       extends [mscorlib]System.Object
{
} // end of class Program
```

Intrinsic CTS Data Types

- In .NET, all primitive data types (Boolean, Byte, Char, DateTime, Decimal, Double, Int16, Int32, Int64, SByte, Single, UInt16, UInt32, and UInt64) are defined as structures

- More about CTS:
  - https://docs.microsoft.com/en-us/dotnet/standard/base-types/common-type-system

Core C# Programming

The Anatomy of a Simple C# Program

- DEMO
  - *Code Labs Up and Running* Revisited

- C# is a case-sensitive programming language
  - **C# keywords** are lowercase (e.g., public, lock, class, dynamic)
  - **Namespaces, types, and member names** begin (by convention) with an initial capital letter and have capitalized the first letter of any embedded words (e.g., System.Data.SqlClient)
  - Produces **compiler error** regarding “undefined symbols”

The Anatomy of a Simple C# Program

- static
    - static members are scoped to the class level (rather than the
    - object level)

- void
    - Does not explicitly define a return value

- Main
    - Entry-point of the application

System.Console Class

- Console User Interface (CUI) may not be as interesting as a Graphical User Interface (GUI) or Web Application
    - However, allows us to keep focused on the syntax of C# and the core aspects of the .NET platform
- *WriteLine()* pumps a text string (including a carriage return) to the output stream
- *ReadLine*() allows you to receive information from the input stream up until the Enter key is pressed

Formatting Console Output

```
// Prints: 20, 10, 30
Console.WriteLine("{1}, {0}, {2}", 10, 20, 30);
```

Formatting Numerical Data

```
Console.WriteLine("{0:<StringFormatCharacter>}", 99999);
```

String Format Character	Meaning
C or c	Used to format currency. By default, the flag will prefix the local cultural symbol (a dollar sign [$] for U.S. English).
D or d	Used to format decimal numbers. This flag may also specify the minimum number of digits used to pad the value.
E or e	Used for exponential notation. Casing controls whether the exponential constant is uppercase (E) or lowercase (e).
F or f	Used for fixed-point formatting. This flag may also specify the minimum number of digits used to pad the value
G or g	Stands for *general*. This character can be used to format a number to fixed or exponential format.
N or n	Used for basic numerical formatting (with commas)
X or x	Used for hexadecimal formatting

System Data Types and Corresponding C# Keywords

C#	CLS ?	System Type	Range	Meaning
bool	Yes	System.Boolean	True or False	Represents Truth of Falsity
sbyte	No	System.SByte	-128 to 127	Signed 8-bit number
byte	Yes	System.Byte	0 to 255	Unsigned 8-bit number
short	Yes	System.Int16	-32768 to 32767	Signed 16-bit number
ushort	No	System.UInt16	0 to 65535	Unsigned 16-bit number
int	Yes	System.Int32	-2147483648 to 2147483647	Signed 32-bit number
uint	No	System.UInt32	0 to 4294967295	Unsigned 32-bit number

System Data Types and Corresponding C# Keywords

C#	CLS ?	System Type	Range	Meaning
long	Yes	System.Int64	-9223372036854775 to 9223372036854775807	Signed 64-bit number
ulong	No	System.UInt64	0 to 18446744073709551615	Unsigned 64-bit number
char	Yes	System.Char	U+0000 to U+ffff	Single 16-bit Unicode Character
float	Yes	System.Single	$-3.4\text{x}10^{38}$ to $+3.4\text{x}10^{38}$	32-bit signed number
double	Yes	System.Double	$5.0 \text{ x } 10^{-324}$ to $1.7 \text{ x } 10^{308}$	64-bit signed number
decimal	Yes	System.Decimal	$(-7.9\text{x}10^{28}$ to $7.9 \text{ x } 10^{28})/10^{0 \text{ to } 28}$	128-bit signed number
string	Yes	System.String	Limited by system memory	Represents a set of Unicode characters
Object	Yes	System.Object	Can store any data type in an object variable	The base class of all types in the .NET universe

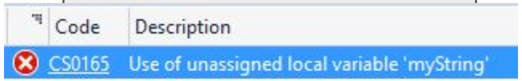
Variable Declaration and Initialization

- Local variable
  - data type followed by the variable's name

- E.g.:
  - int myInt;
  - string myString;

- Initialization
  - int myInt = 0;
  - string myString = "Code Labs";

Trying to Write to console *myString* variable outputs a compiler error



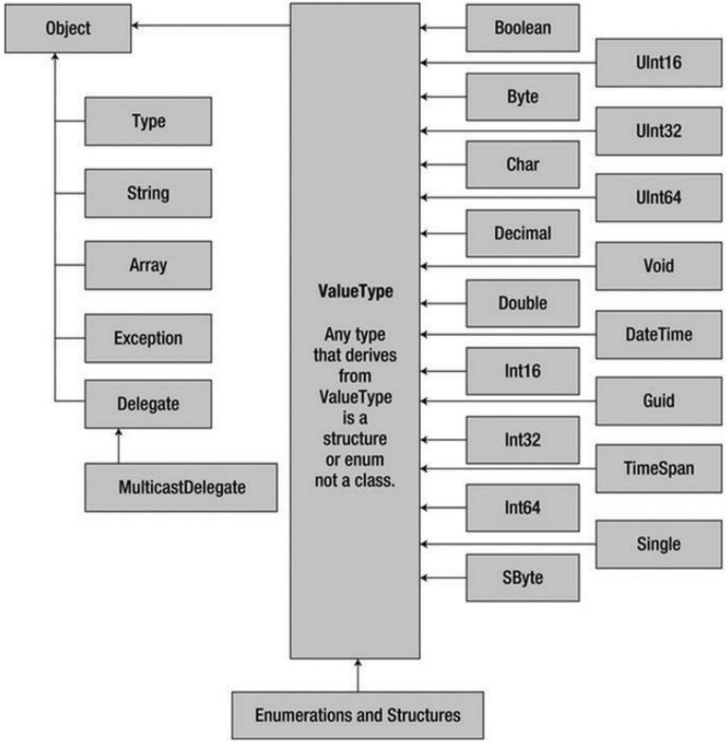
Code	Description
CS0165	Use of unassigned local variable 'myString'

Intrinsic Data Types and the new Operator

- All intrinsic data types support what is known as a *default constructor*
    - bool variables are set to false
    - Numeric data is set to 0
    - char variables are set to a single empty character
    - BigInteger variables are set to 0
    - DateTime variables are set to 1/1/0001 12:00:00 AM
    - Object references (including strings) are set to null

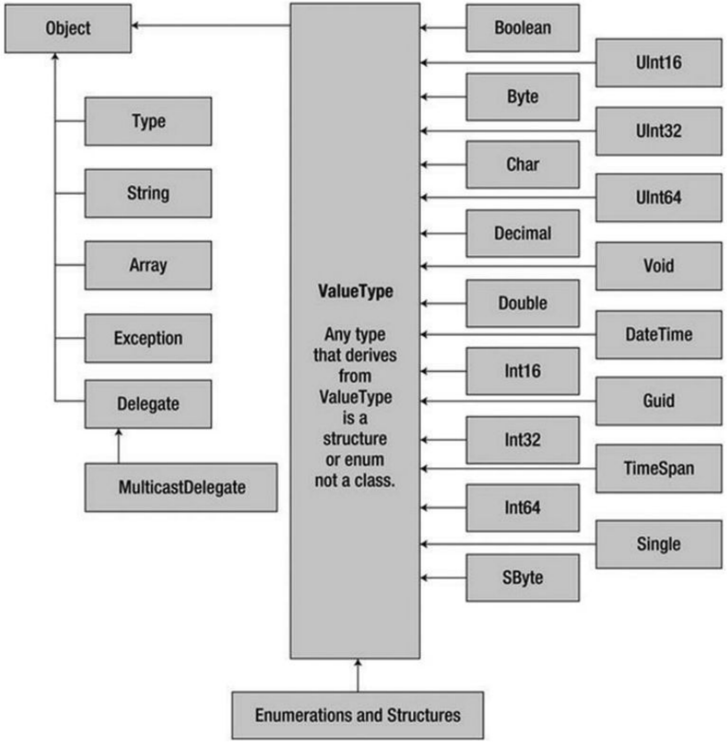
The Data Type Class Hierarchy

- Note that even the primitive .NET data types are arranged in a class hierarchy
- Each type ultimately derives from System.Object
    - defines a set of methods (e.g., ToString(), Equals(), GetHashCode())
- Numerical data types derive from a class named System.ValueType
    - Allocated on the stack
    - Have a predictable lifetime
    - Efficient



The Data Type Class Hierarchy

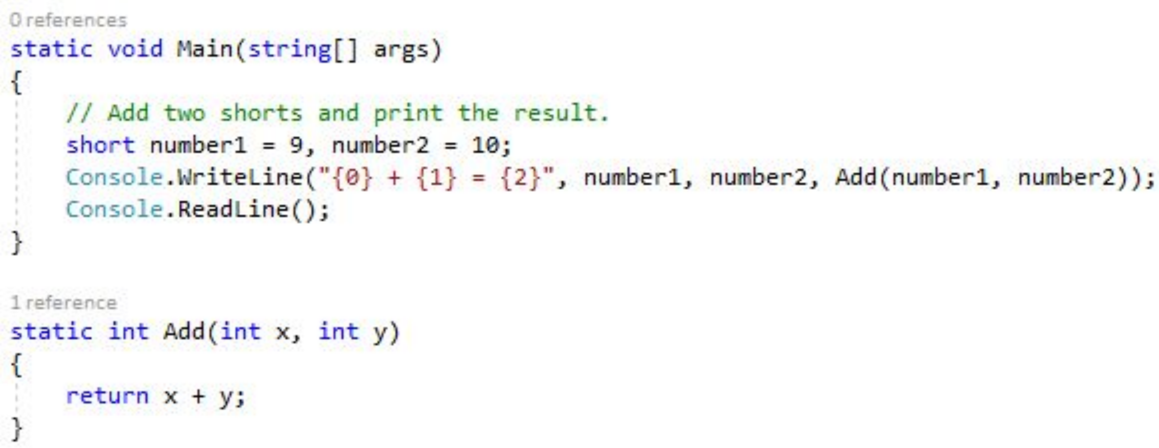
- Types that **do not have** System.ValueType in their inheritance chain
  - Are allocated on the garbage-collected heap



Members of Data Types

- Practice:
  - Explore, using intellisense and documentation:
    - What is the MaxValue and MinValue of an Integer
    - How to check if a char is a letter or a digit
    - How to add/subtract days from a date
    - Some string manipulation, including concatenation, verbatim, string equality and inequality

Narrowing and Widening Data Type Conversions



```
0 references
static void Main(string[] args)
{
    // Add two shorts and print the result.
    short number1 = 9, number2 = 10;
    Console.WriteLine("{0} + {1} = {2}", number1, number2, Add(number1, number2));
    Console.ReadLine();
}

1 reference
static int Add(int x, int y)
{
    return x + y;
}
```

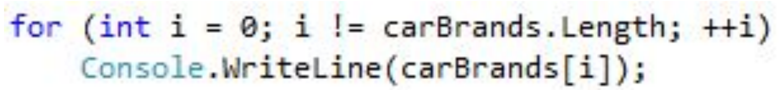
C# Iteration Constructs

- C# provides the following four iteration constructs
  - for loop
  - foreach/in loop
  - while loop
  - do/while loop

- Consider
  - string[] carBrands = {"Ford", "BMW", "Renault", "Honda" };

C# Iteration Constructs - for

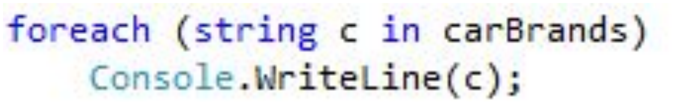
- Initial condition
    - Int i = 0;
- Condition
    - i != carBrands.Length
- Step
    - ++i



```
for (int i = 0; i != carBrands.Length; ++i)
    Console.WriteLine(carBrands[i]);
```

C# Iteration Constructs - foreach

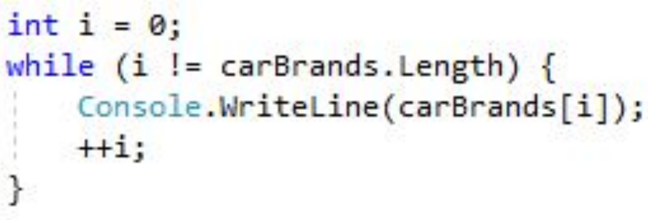
- foreach string in the string[]



```
foreach (string c in carBrands)
    Console.WriteLine(c);
```

C# Iteration Constructs - while

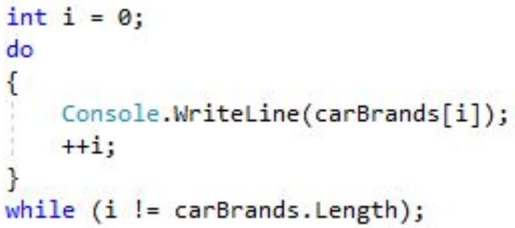
- Initial condition
  - Int i = 0;
- Condition
  - i != carBrands.Length
- Step
  - ++i



```
int i = 0;
while (i != carBrands.Length) {
    Console.WriteLine(carBrands[i]);
    ++i;
}
```

C# Iteration Constructs - do/while

- Initial condition
    - Int i = 0;
- Condition
    - i != carBrands.Length
- Step
    - ++i



```
int i = 0;
do
{
    Console.WriteLine(carBrands[i]);
    ++i;
}
while (i != carBrands.Length);
```

C# Iteration Constructs

- All very similar. When do I use each one?
  - for: we know both the conditions and step
  - foreach: avoid using indices
  - while: we don’t know the step, the step is different depending on conditions. Focus on the condition.
  - do/while: do at least one time, even if condition is not true

Decision Constructs and the Relational/Equality Operators

- if/else statement
  - Control program execution based on boolean conditions
  - The condition is true, execute the first code block
  - Execute the second code block otherwise

Decision Constructs and the Relational/Equality Operators

C# Equality/Relational Operator	Example Usage	Meaning
==	if(age == 30)	Returns *true* if age is 30
!=	if(myStr != “abc”)	Returns *true* if myStr is not “abc”
<	if(bonus < 2000)	Returns *true* if bonus verifies condition
>	if(bonus > 2000)	
<=	if(bonus <= 2000)	
>=	if(bonus >= 2000)	

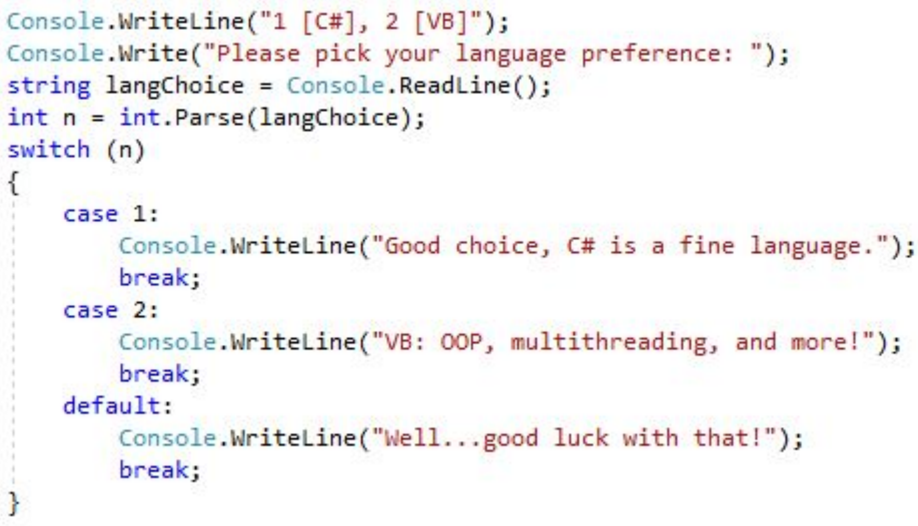
Conditional Operators

- if statement may be composed of complex expressions

Operator	Example	Meaning
&&	if(age == 30 && name == “John”)	AND operator. Returns *true* if all expressions are true.
||	if(age == 30 || name == “John”)	OR operator. Returns *true* if at leats one expression are true. If one is, others are not evaluated.
!	if(!myBool)	NOT operator. Returns true if false or false if true

The switch Statement

- Allows you to handle program flow based on a predefined set of choices



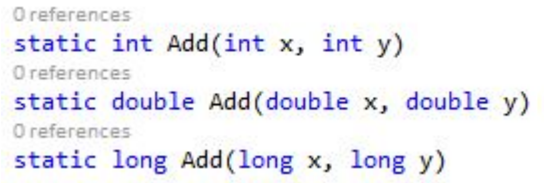
```
Console.WriteLine("1 [C#], 2 [VB]");
Console.Write("Please pick your language preference: ");
string langChoice = Console.ReadLine();
int n = int.Parse(langChoice);
switch (n)
{
    case 1:
        Console.WriteLine("Good choice, C# is a fine language.");
        break;
    case 2:
        Console.WriteLine("VB: OOP, multithreading, and more!");
        break;
    default:
        Console.WriteLine("Well...good luck with that!");
        break;
}
```

Parameter Modifiers

Parameter Modifier	Meaning
(None)	If a parameter is not marked with a parameter modifier, it is assumed to be passed by value, meaning the called method receives a copy of the original data.
out	Output parameters must be assigned by the method being called and, therefore, are passed by reference. If the called method fails to assign output parameters, you are issued a compiler error.
ref	The value is initially assigned by the caller and may be optionally modified by the called method (as the data is also passed by reference). No compiler error is generated if the called method fails to assign a ref parameter.
params	This parameter modifier allows you to send in a variable number of arguments as a single logical parameter. A method can have only a single params modifier, and it must be the final parameter of the method. In reality, you might not need to use the params modifier all too often; however, be aware that numerous methods within the base class libraries do make use of this C# language feature.

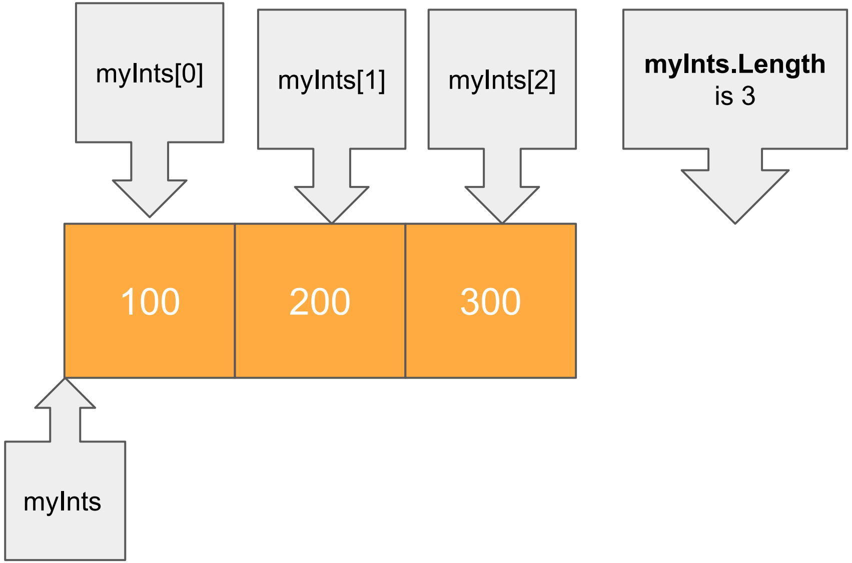
Understanding Method Overloading

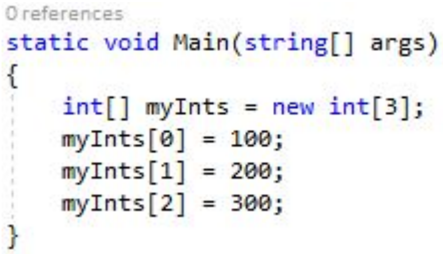
- Define a set of identically named methods that differ by the number (or type) of parameters, the method in question is said to be overloaded



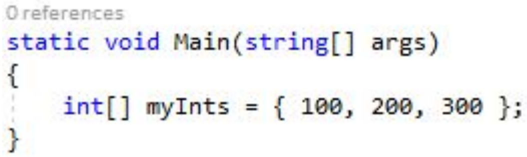
```
0 references
static int Add(int x, int y)
0 references
static double Add(double x, double y)
0 references
static long Add(long x, long y)
```

Understanding C# Arrays



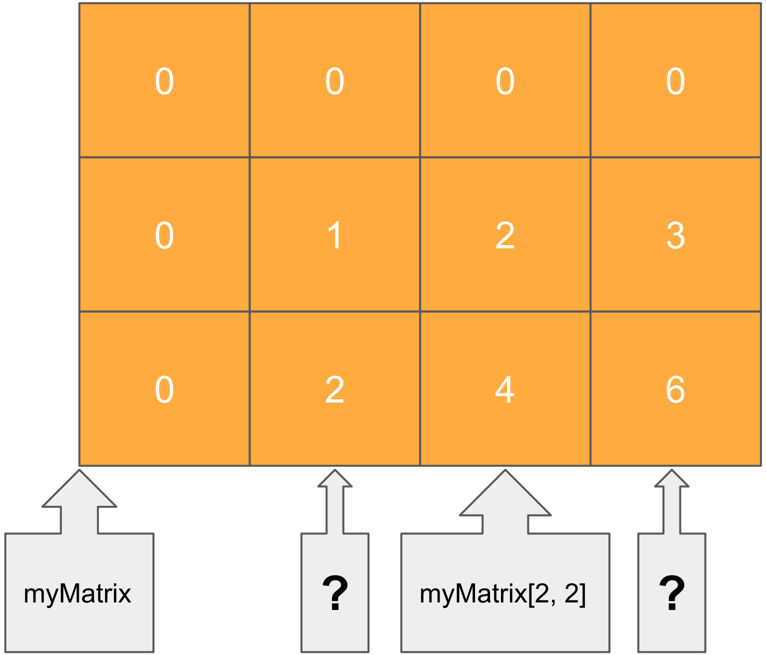


```
0 references
static void Main(string[] args)
{
    int[] myInts = new int[3];
    myInts[0] = 100;
    myInts[1] = 200;
    myInts[2] = 300;
}
```



```
0 references
static void Main(string[] args)
{
    int[] myInts = { 100, 200, 300 };
}
```

Working with Multidimensional Arrays



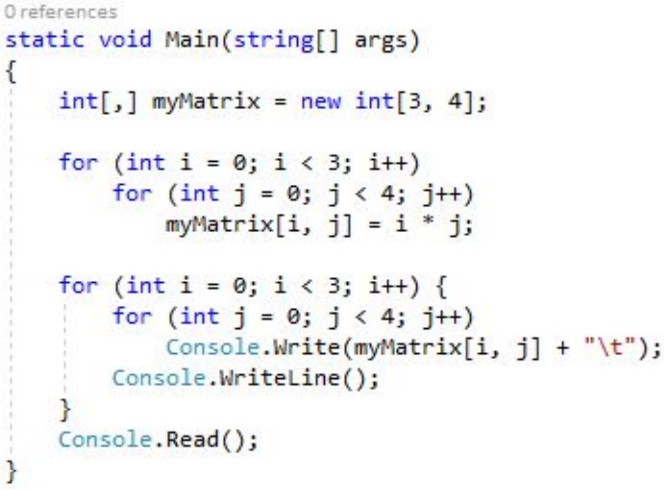
0	0	0	0
0	1	2	3
0	2	4	6

myMatrix

?

myMatrix[2, 2]

?

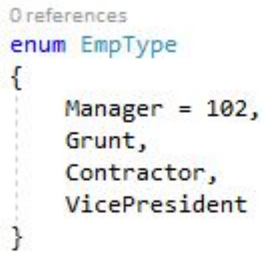


```
0 references
static void Main(string[] args)
{
    int[,] myMatrix = new int[3, 4];

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 4; j++)
            myMatrix[i, j] = i * j;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++)
            Console.Write(myMatrix[i, j] + "\t");
        Console.WriteLine();
    }
    Console.Read();
}
```

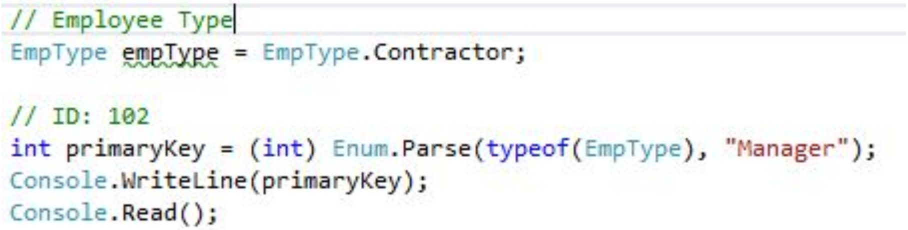
More info: https://msdn.microsoft.com/en-us/library/system.array(v=vs.110).aspx

Understanding the enum Type

- When building a system, it is often convenient to create a set of symbolic names that map to known numerical values
    - E.g., Mapping database PK into a *dropdown list*



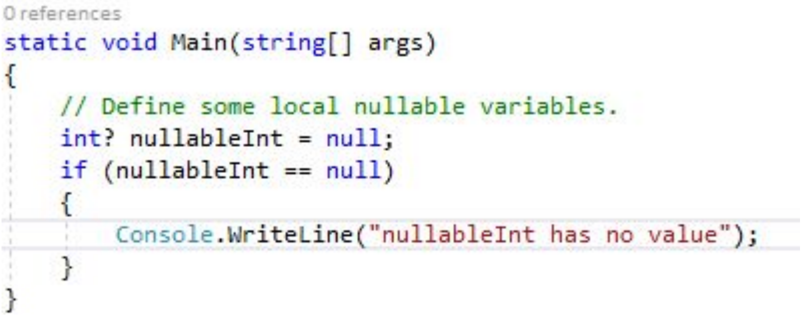
```
0 references
enum EmpType
{
    Manager = 102,
    Grunt,
    Contractor,
    VicePresident
}
```



```
// Employee Type
EmpType empType = EmpType.Contractor;

// ID: 102
int primaryKey = (int) Enum.Parse(typeof(EmpType), "Manager");
Console.WriteLine(primaryKey);
Console.Read();
```

Understanding C# Nullable Types

- How to distinguish between the integer 0 and the absence of value?
    - Nullable Types are testable with *null*

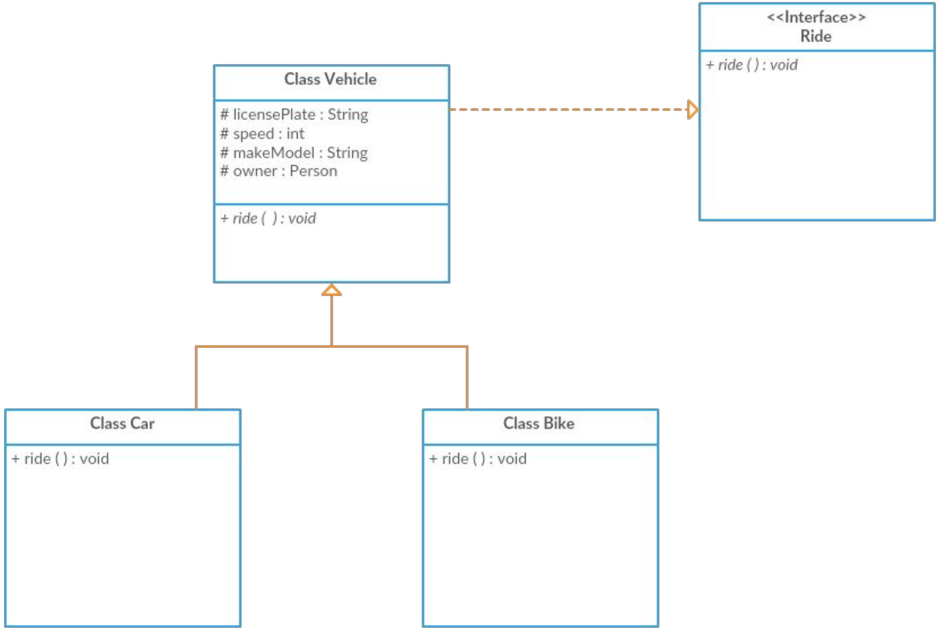


```
0 references
static void Main(string[] args)
{
    // Define some local nullable variables.
    int? nullableInt = null;
    if (nullableInt == null)
    {
        Console.WriteLine("nullableInt has no value");
    }
}
```

Object Oriented Programming (OOP) with C#

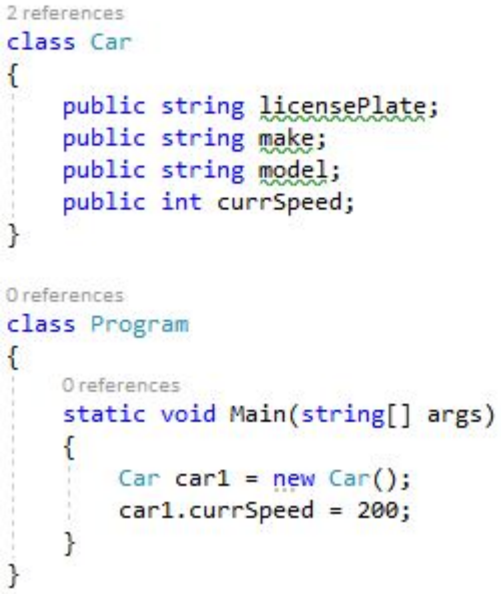
- *Encapsulation, Inheritance and Polymorphism*
- *Interfaces*
- *Structured Exception Handling*

My First Class, Constructors and Properties



My First Class, Constructors and Properties

- Create a class
- Difference between public, private and protected
- Class Vehicle as Super Class
  - Bike, Car and Boat
  - Polymorphic Ride
- *This* keyword



```
2 references
class Car
{
    public string licensePlate;
    public string make;
    public string model;
    public int currSpeed;
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Car car1 = new Car();
        car1.currSpeed = 200;
    }
}
```

OOP Pillars

- Encapsulation
  - language’s ability to hide unnecessary implementation details from the object user
- Inheritance
  - language’s ability to allow you to build new class definitions based on existing class definitions
- Polymorphism
  - language’s ability to treat related objects in a similar manner

Interfaces

- A named set of abstract members

- An interface expresses a behavior that a given class or structure may choose to support
  - E.g., IDbConnection

- Interface Types vs. Abstract Base Classes

Structured Exception Handling

- Bugs
  - These are, simply put, errors made by the programmer
- User errors
  - User errors, on the other hand, are typically caused by the individual running your application, rather than by those who created it
- Exceptions
  - Exceptions are typically regarded as runtime anomalies that are difficult, if not impossible, to account for while programming your application.

The Building Blocks of .NET Exception Handling

- A class type that represents the details of the exception

- A member that throws an instance of the exception class to the caller under the correct circumstances

- A block of code on the caller's side that invokes the exception-prone member

- A block of code on the caller's side that will process (or catch) the exception, should it occur

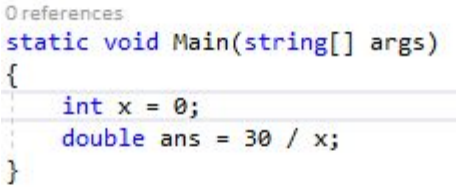
The Building Blocks of .NET Exception Handling

- C# offers five keywords
  - Try, catch, throw, finally and when

- All exceptions extend System.Exception

Example

- What happens when you execute the following program?



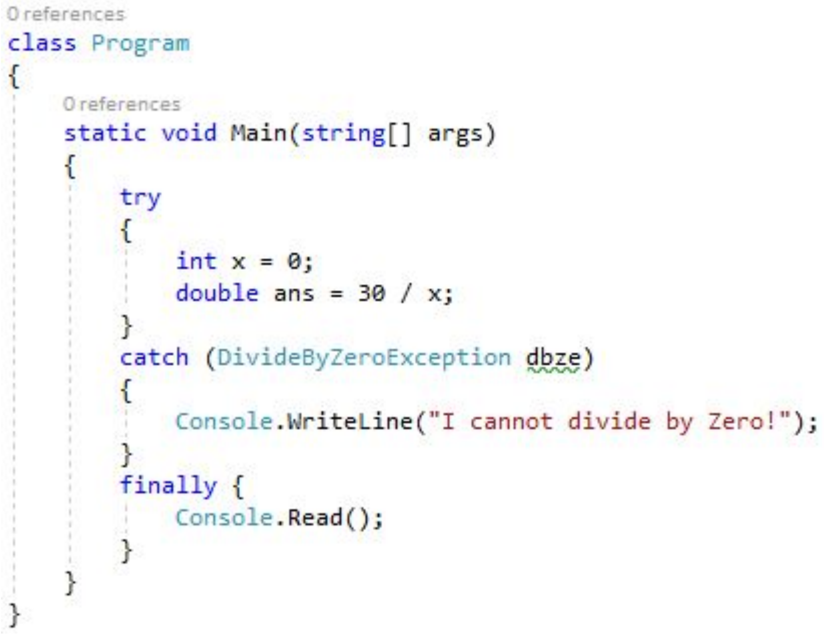
```
static void Main(string[] args)
{
    int x = 0;
    double ans = 30 / x;
}
```

Example

- What is wrong with this code?

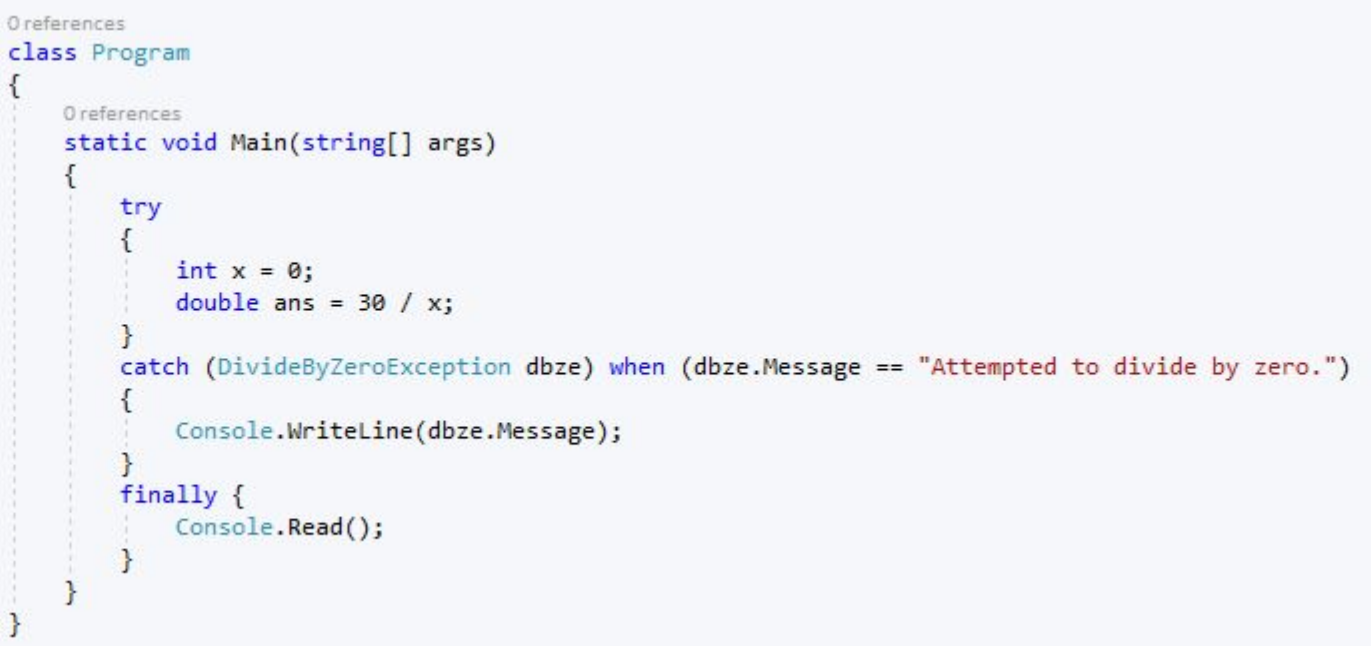
- Should this particular exception be treated as exception?

- Should finally be blocking?



```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        try
        {
            int x = 0;
            double ans = 30 / x;
        }
        catch (DivideByZeroException dbze)
        {
            Console.WriteLine("I cannot divide by Zero!");
        }
        finally {
            Console.Read();
        }
    }
}
```

Example



```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        try
        {
            int x = 0;
            double ans = 30 / x;
        }
        catch (DivideByZeroException dbze) when (dbze.Message == "Attempted to divide by zero.")
        {
            Console.WriteLine(dbze.Message);
        }
        finally {
            Console.Read();
        }
    }
}
```

Notes

- Do not catch all Exceptions
    - If you need to, be sure to understand this will filter other exception catch blocks

- Create your own Exceptions if needed

Advanced C#

- *Collections*
- *Delegates, Events and Lambda Expressions*
- *Understanding Object Lifetime*

Collections

- The System.Collections namespace contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hash tables and dictionaries
  - https://docs.microsoft.com/en-us/dotnet/standard/collections/

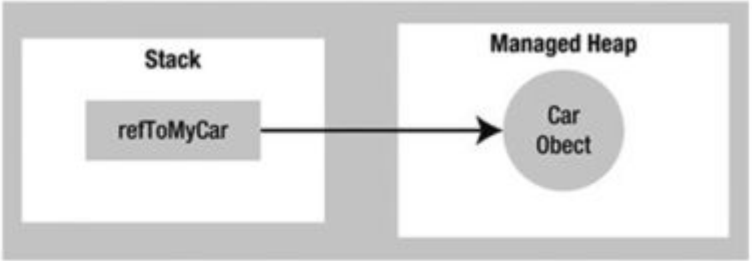
Delegates, Events and Lambda Expressions

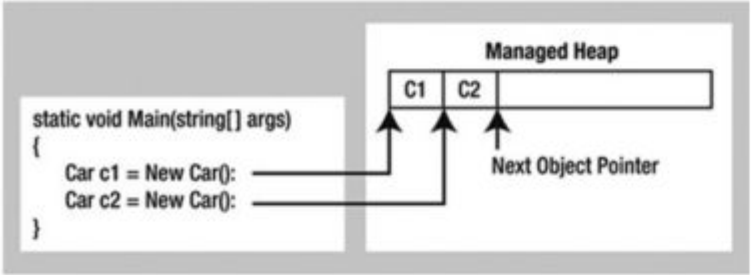
- A **delegate** is an object which represents a method
    - When the delegate is invoked, the corresponding method is invoked
- **Events** are a specialized delegate type primarily used for message or notification passing

- **Anonymous methods** are expressions that can be converted to delegates

Understanding Object Lifetime

- Garbage Collector
  - The System.GC Type
  - https://msdn.microsoft.com/en-us/library/system.gc(v=vs.110).aspx
  - https://docs.microsoft.com/en-gb/visualstudio/profiling/understanding-memory-allocation-and-object-lifetime-data-values





```
static void Main(string[] args)
{
    Car c1 = New Car():
    Car c2 = New Car():
}
```

.NET Base Class Libraries

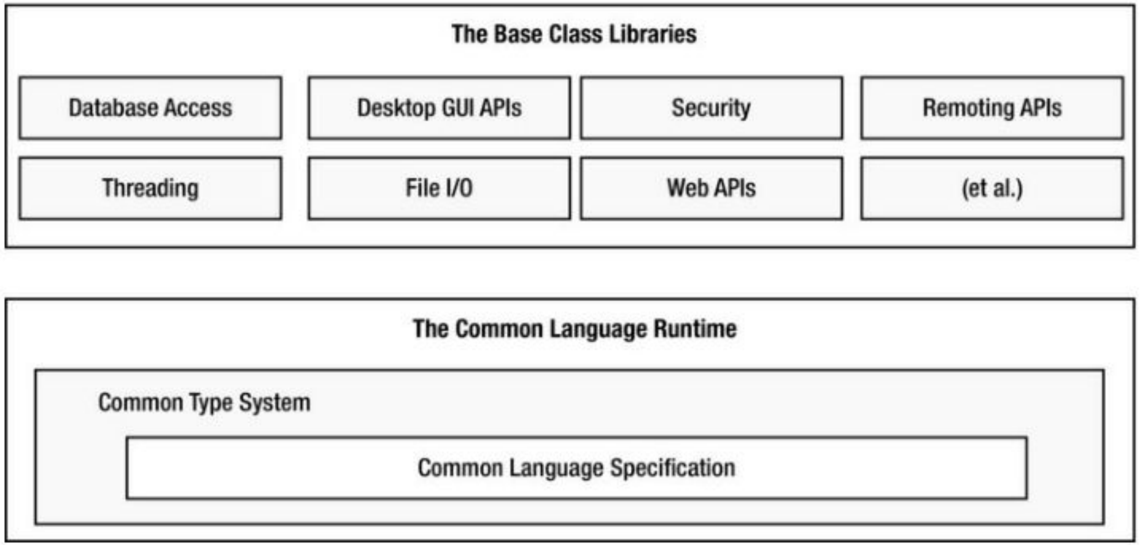
- *Multithreaded, Parallel and Async Programming*
- *File I/O*

The Role of the Base Class Libraries

- Encapsulate various primitives such as
  - Threads,
  - File input/output (I/O),
  - Graphical rendering systems, and
  - Interaction with various external hardware devices

- Provides support for a number of services required by most real-world applications.

The Role of the Base Class Libraries



The Base Class Libraries

Database Access

Desktop GUI APIs

Security

Remoting APIs

Threading

File I/O

Web APIs

(et al.)

The Common Language Runtime

Common Type System

Common Language Specification

.NET Base Class Libraries

- File and Stream I/0
  - https://docs.microsoft.com/en-us/dotnet/standard/io/index

- Async Programming:
  - https://docs.microsoft.com/en-us/dotnet/csharp/async

- Multithreaded Applications:
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/threading/multithreaded-applications

Where can I find information?

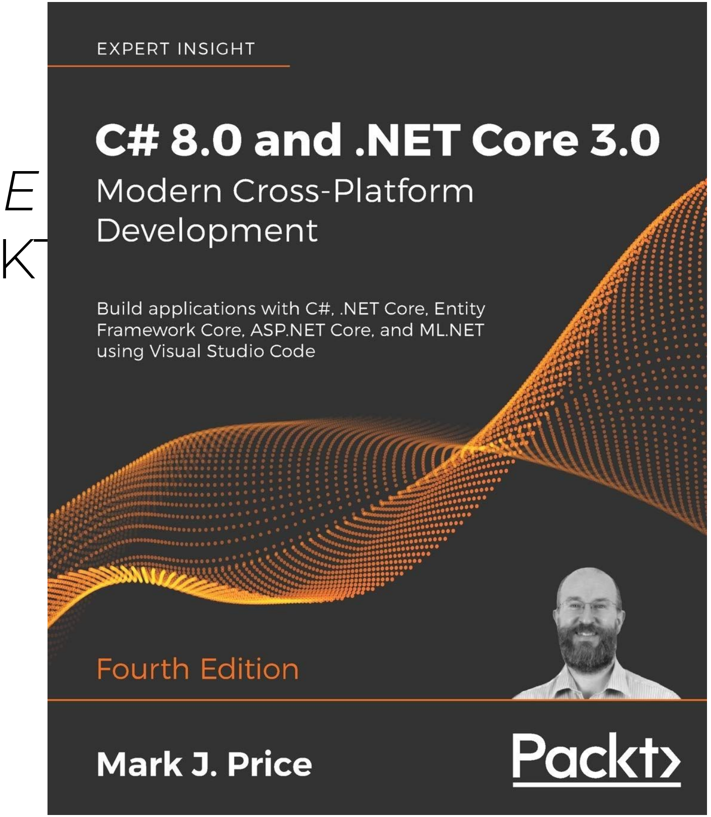
- Microsoft’s Developers Network
    - https://msdn.microsoft.com/library
- C# Reference
    - https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/index
- .NET Framework Guide
    - https://docs.microsoft.com/en-us/dotnet/framework/index

Resources and Tools

- Code online, everywhere: https://dotnetfiddle.net/
- News: http://blog.cwa.me.uk/tags/morning-brew/
- https://stackoverflow.com/
- https://github.com/dotnet/
- https://docs.microsoft.com/en-us/dotnet/
- https://pt.lmgtfy.com/?q=dotnet&s=l

References

- Price, Mark J. "*C# 8.0 AND .NET CORE* *Cross-Platform Development*", PACK
- .NET Core documentation
- C# Documentation
- https://github.com/dotnet/core



EXPERT INSIGHT

C# 8.0 and .NET Core 3.0

Modern Cross-Platform Development

Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code

Fourth Edition

Mark J. Price

Packt>