**C#**

(Learning from courses in PluralSight’s *C# Development Fundamentals* path, Microsoft Docs’ *C# Tutorials*)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Environment** | | | | | | | | | | |
| .NET Core SDK | - A set of libraries and tools that allow developers to create NET Core applications and libraries (build and run)  - Takes program code and translates it to tell the computer what to do  - Completely open-sourced and runs on many OSes (different from .NET Framework – Windows only)  - Is a runtime  - Common Language Runtime (CLR)  - Provides a space to run code  - Is a library  - Framework Class Library (FCL) | | | | | | | | |  |
| Folders | | | | | |  | | | |
| |gradebook  | src  | GradeBook  | test | | | | | | Example of a project structure | | | |
| src | | | | | | - Can have many (project) directories,  - Each representing a project that is part of a system (an individual application/component of a part of a larger application) | | | |
| obj | | | | | | Contains some temporary files created during the restore and build process  - In main folder container csproj file  - Can be safely deleted | | | |
| bin | | | | | | Stands for “binary”  - In main folder container csproj file  - Can be safely deleted | | | |
| .vscode | | | | | | - Files inside control what VS Code does when debugging or building project | | | |
| Run on Terminal | | | | | | Explanation | | | |
| dotnet --info | | | | | | Run on terminal  - Gives version of .NET SDK (tool using to target .NET development), .NET runtime (framework/library being used to execute .NET program) being used | | | |
| dotnet --help | | | | | | Lists available SDK commands | | | |
| dotnet new [app shortname] | | | | | | Uses a template to create a new application  - Find some examples of templates by running dotnet new  - E.g. dotnet new console creates a new console (CLI) application  - .csproj extension is for C# projects | | | |
| dotnet **run**  # If not executing in proj dir  dotnet run --project src/appName | | | | | | Execute code in *project directory*  - This is the folder which has the .csproj file  - E.g. appName  - Under-the-hood, actually runs dotnet restore + dotnet build | | | |
| dotnet restore | | | | | | - Looks through csproj file for external depencies in code | | | |
| dotnet build | | | | | | - Compile source code  - Analyses C# code/multiple files and translate it into an efficient binary format that is faster to execute  - Outputs a .dll file | | | |
| type “FileName.txt” | | | | | | Print contents of file onto console | | | |
| Terminal Examples | | | | | |  | | | |
| dotnet run Scott | | | | | | Pass a parameter to .NET CLI  - “Scott” becomes parameter **not for application** | | | |
| dotnet run **--** Scott | | | | | | Pass a parameter to application  - Pass “--” to separate .NET CLI and application parameters | | | |
| dotnet run **--project** src/GradeBook/GradeBook.csproj | | | | | | Run a **specific** project from the most top-level email | | | |
| NuGet | - Package system in .NET  - Allows to taking advantage of features and functionality in libraries written by others (i.e. “NuGet packages”)  - Made available through a package feed on the internet  - Similar to NodeJS’s npm and Python’s pip | | | | | | | | |  |
| - Visit *nuget* website (nuget.org) to find command to run to use a NuGet package (package reference)  - Look at .csproj file in see what packages are being referenced currently in project | | | | | | | | |  |
| SDK | - Stands for “Software Development Kit” (or known as “devkit”)  - Provides a set of tools, libraries, relevant documentation, code samples, processes and/or guides  - Help developers create software applications on a specific platform  - Different from API (“Application Programming Interface”), API designed for interaction between applications while SDK designed for *creation* of applications/designing of functions  - Usually includes APIs (but not the other way round) | | | | | | | | |  |
| File with .dll extension | - Is an **assembly**  - Represents an output of a complier  - Does not mean “Dynamic Link Library”  - Stored in the bin (binary) folder  - Also in Debug folder (file is a debug build of application | | | | | | | | | Example (run from project directory) dotnet bin\Debug\netcoreapp3.1\GradeBook.dll  - Similar Java’s  java Duke.jar |
| VisualStudio Code |  | | | | | | | | |  |
| Run on Terminal | | | | | | | | Example | |
| code . | | | | | | | | - When VS Code is installed, option to add code to path is given  - Allows VS code to be launched from anywhere in the file system by typing code  - Dot (.) indicates to start editor in the current directory in  - Usually the most top-level folder | |
| On VS Code | | | | | | | |  | |
| - Prompt that assets and debug are missing from the project | | | | | | | | - Click Yes  - Makes it easier to run and debug in VS Code  - Make sure that extension C# is installed in VS Code | |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
| **Debugging** | | | | | | | | | | |
| Set a breakpoint | | | Beside the code line, place a red dot | | | | | | |  |
| Run program in debug mode | | | Method 1:  - Menu Run > Start Debugging  Method 2:  - Shortcut: F5 | | | | | | |  |
| Exit debug mode | | | Method 1:  - Menu Run > Stop Debugging  Method 2:  - Shortcut: Shift + F5 | | | | | | |  |
| Pass in parameters while in Debug mode | | | 1. View launch.json file in .vscode folder  2. Look under “configurations” key  3. Modify value of array under “args” key  - Value of “args” is the arguments array that the debugger will pass to application) | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
|  |  | | | | | | | | |  |
| **Unit Testing** | | | | | | | | | | |
| - Prove or verify that code is working correctly  - Also test edge conditions  - Located in test folder  - **Test file name should match the class name** | | | | | | | | | | |
| XUnit | - Test library popular in .NET  - Not part of .NET Core, relies on libraries other than that from .NET Core | | | | | | | | | Example:  using System;  using Xunit;  namespace Gradebook.Tests  {  public class UnitTest1  {  [Fact]  public void Test1()  {  …  }  }  } |
| Place executable statements inside a method to be invoked  - Method has to be a member of a type  - Discovery and test execution happens when *test runner* runs | | | | | | | | |  |
| Test runner | - .NET CLI includes one test runner  - VS Code has test runners through extensions | | | | | | | | |  |
| Attribute | - Attached to a method  - A little piece of data attached to the symbols that follows it  - When executing test project, XUnit will involve methods with a [Fact] attribute attached to test if it passes or fails  - API for tests provided by Assert class in xUnit namespace | | | | | | | | | Examples:  - [Fact] represents method is a unit test |
| Folders | | | |  | | | | | |
| |gradebook  | src  | …  | test  | GradeBook.Tests | | | | Example of a project structure  - Folder to do xunit testing (usually named with AppName**.Tests**) | | | | | |
|  | | | |  | | | | | |
|  | | | |  | | | | | |
|  | | | |  | | | | | |
| Terminal Examples | | | |  | | | | | |
| dotnet test | | | | - Run in the directory with the test files (files with .cs extensions)  - Execute all unit tests found (i.e. methods with [Fact] attached to it) | | | | | |
| dotnet new xunit | | | | Uses template to create new testing project | | | | | |
| dotnet **add package** xunit | | | | Add reference to xunit package | | | | | |
| dotnet **add reference** ..\..\src\  GradeBook\  GradeBook.csproj | | | | Add a reference to a project from a project  - Reference to the .csproj file | | | | | |
| Structure of a unit test | Section | | |  | | | | | | |
| 1. ARRANGE | | | - Where all test data are put together  - Data and values to used are arranged | | | | | | |
| 2. ACTUAL | | | - Where a method is invoked  - Produces an actual result | | | | | | |
| 3. ASSERT | | | - Test where actual results matches expected | | | | | | |
|  | | | | | | | | |  |
| Assert class |  | | | | | | | | |  |
| Code | | | |  | | | | | |
| Assert**.Equal**(expected, actual) | | | |  | | | | | |
| Assert.**Same**(obj1, obj2) | | | | Check that two objects are of the same instance (same reference pointer) | | | | | |
| Assert.**NotSame**(obj1, obj2) | | | | Check that two objects are **not** the same instance | | | | | |
| Assert.**True**(  Object.ReferenceEquals(obj1, obj2) | | | | Same function as Assert.Same() | | | | | |
| Assert.True(obj1, obj2) | | | |  | | | | | |
| Solution file | - A file that keeps track of multiple projects  - Understood by VS Code, .NET CLI  - Allows all mentioned projects to be loaded for editing in VS Code or from command line  - Single command can be executed to apply to all projects in solution (e.g. dotnet build) | | | | | | | | |  |
| - Create in the most top level folder (even higher than the one containing the .csproj file)  - Has the extension .sln | | | | | | | | |  |
| Run on terminal | | | | |  | | | | |
| dotnet new sln | | | | | Creates a new solution file (from a template) | | | | |
| **dotnet sln** add src/GradeBook/GradeBook.csproj | | | | |  | | | | |
| dotnet build  dotnet test | | | | | - Run in most top-level folder  - Looks for a solution directory  - Will execute command on all projects mentioned in the solutions file | | | | |
|  |  | | | | | | | | |  |
| **Other features** | | | | | | | | | | |
| Garbage Collection | | | - Don’t need to tell runtime to delete object or free up the memory when object is no longer needed  - .NET runtime will keep track of all objects, variable allocated and created, all the fields and pointers  - .NET runtime know when there is an object in memory with no variables and fields pointing to it | | | | | | | |
| Method Overloading | | | - Overload methods with the same method name but different method **signature** (e.g. parameter, return type etc) | | | | | | | |
| Reference Type | | | | | | | | Value Type | | |
| - **Any class written** (by you, or provided by .NET/NuGet)  - Variables are store a value representing a memory location  - At this address/location, is the object being referenced to  - From this object, .NET runtime can find the attributes with the object (these fields are references themselves)  - Pointer from variable to referenced object | | | | | | | | - Refer to special types in .NET  - E.g. numbers (integers, doubles, floats), can be created by user too, a struct (i.e. special type)  - Memory creates space for variable but it doesn’t store a reference but the value itself  - Always **pass by value** (take not when writing unit tests) | | |
| var b = new Book(“Grades”); | | | | | | | | var x = 3; | | |
| **Pass-by-reference** | | | | | | | | **Pass-by-value**  - Taking value inside of variable and copying it  - Happens when invoking a method (parameters are passed not by reference) | | |
| Defining Properties | | | - Control access of variables and methods | | | | | | |  |
| Refactoring | | | To change a method name across all files referencing it  1. Right-click on method name  2. Select ‘Rename Symbol’  3. Type in new method name | | | | | | |  |
|  | | |  | | | | | | |  |
|  | | |  | | | | | | |  |
|  | | |  | | | | | | |  |
|  | | |  | | | | | | |  |
| **Type Safety** | | | | | | | | | | |
| - Prevent objects of one type from peeking into the memory assigned for another object  - Write safe code to prevent data loss during conversion of one type to another  - Prevents casting an object of a type to another  - For example, if using out or ref in method parameter and when calling method, argument does not match parameter; compiler will now allow code to compile  - In essence, you cannot use the same reference (memory space) and point it to a new object of a different type  - Nor can you try to create a new object by type-casting an object of a different type | | | | | | | | | | Example:  public class SomeType {}  private void ChangeValue(out object par)  {  par = new String(‘x’, 10);  }  SomeType obj = new SomeType();  // Compile time error  ChangeValue(out obj);  // No error (caller can have argument type that derives from parameter type)  ChangeValue(obj); |
|  | |  | | | | | | | |  |
|  | |  | | | | | | | |  |
|  | |  | | | | | | | |  |
|  | |  | | | | | | | |  |
|  | |  | | | | | | | |  |
|  | |  | | | | | | | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Code | |  | | | | | | | | | | | | Example |
| **//** this is a comment in C# | | Comment | | | | | | | | | | | |  |
| **;** | | End statements with a **semi-colon**  - Like Java  - Allows the compiler to understand where one statement ends and the next begins | | | | | | | | | | | |  |
| **var** x = …; | | *Implicit typing*  - Need to have equals to result  - Let compiler guess the type | | | | | | | | | | | | // double  var x = 34.1; |
| PascalCase | | Convention for naming methods, types, properties, public members | | | | | | | | | | | |  |
| camelCase | | Convention for naming local variables and parameters | | | | | | | | | | | |  |
| object | | Lowest base type in .NET framework | | | | | | | | | | | |  |
| index**++**; | | Add one | | | | | | | | | | | |  |
| **when** | | Keyword that allows condition to only be evaluated at runtime | | | | | | | | | | | |  |
| **&&** (AND),  **||** (OR),  == (EQUAL) | | Boolean operators to use in conditions | | | | | | | | | | | |  |
| **nameof**(obj) | | Returns a string representation of object | | | | | | | | | | | |  |
| // Results in a compiler error  var p = new Program();  p.Main(args);  // Program will run indefinitely  Program.Main(args); | | Main method is a class/static method and not an instance method  - Cannot be used like an object | | | | | | | | | | | |  |
| **Inputs and Outputs** | | | | | | | | | | | | | | |
| Console.**WriteLine**(“…”); | | Outputs/prints string to console | | | | | | | | | | | |  |
| Console.**ReadLine**(); | | Reads CLI input as string | | | | | | | | | | | |  |
| double.**Parse**(input); | | Parse input string from CLI into data type *double* | | | | | | | | | | | |  |
| var writer = File.**AppendText**(“FileName.txt”) | | Opens a file with specified file name and extension  - Generally when a file is opened for writing, the file is locked and cannot be opened again | | | | | | | | | | | | - In package System.IO  - Implements IDisposable |
| writer.**WriteLine**(value); | | Writes to a file | | | | | | | | | | | |  |
| writer.**Close**(); | | Close a file  - Clean and free up underlying resource | | | | | | | | | | | | Make sure to always invoke either of these methods after a file is opened (take note of exceptions when writing to the file which may prevent this from happening) |
| writer.**Dispose**(); | | Alternative to .close() | | | | | | | | | | | |
| **using** (var writer = File.AppendText($”{Name}.txt”))  {  writer.WriteLine(grade);  } | | | | Guarantees that dispose will be called on IDisposable object  - Check that object has base class IDisposable by using F12 on the class/method name  - Essentially like a try-finally where it will execute code and definitely close the file | | | | | | | | | |  |
| reader = File.**OpenText**(“FileName.txt”) | | | |  | | | | | | | | | |  |
| reader.**ReadLine**() | | | |  | | | | | | | | | |  |
| **using** (var reader = File.**OpenText**($”{Name}.txt”))  {  reader.ReadLine();  } | | | |  | | | | | | | | | |  |
| **Namespaces** | | | | | | | | | | | | | | |
| - Similar to packages in Java  - Many functions can be defined in the same namespace but only **1 class** per file  - Helps to differentiate between classes named similarly  - Avoids conflicting with types and classes defined by others (by not writing code in the global namespace) | | | | | | | | | | | | | | fille1.cs  namespace Food  {  class Program { … }  }  file2.cs  // Same namespace  namespace Food  {  class Veggie { … }  } |
| using System; | | Using namespaces  - Place at top of file, before any code  - Defined by Microsoft | | | | | | | | | | | |  |
| Identifying namespace of a class | | 1. Hover cursor over class name (e.g. list …)  2. Press CTRL+. | | | | | | | | | | | |  |
| Systems.Collections.Generic | | Contains classes like data structures (e.g. stack, queue, list) | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **Control Statements/Terms** | | | | | | | | | | | | | | |
| **if** (condition1)  {  …;  }  **else if** (condition2)  {  …;  }  else  {  …;  } | | If-else statement | | | | | | | | | | | |  |
| **for** (var i = 0; i < arr.Count; i++)  {  …  } | | For-loop | | | | | | | | | | | |  |
| **while** (cond)  {  …  } | | While-loop | | | | | | | | | | | |  |
| **do**  {  …  } **while** (terminatingCond); | | Do-while loop | | | | | | | | | | | |  |
| **foreach**(**var** value **in** arr)  {  …  } | | For-each statement | | | | | | | | | | | |  |
| **switch**(var)  {  **case** ‘A’:  …  **break**;  case ‘’B’:  …  break;  …  **default:**  …  break;  } | | Switch-statement  - No need curly braces after each case | | | | | | | | | | | | Note:  - If using characters in cases, use **single-quotes** as double-quotes will treat even a single-letter as a string |
| **switch**(var)  {  **case** var d **when** d >= 90:  …  break;  …  } | | Another variation of the switch-statement  - Use when keyword  - Allows for pattern matching (type matches, comparisons etc) | | | | | | | | | | | |  |
| **continue;**  **break;** | | Usual jumping statements | | | | | | | | | | | |  |
| if (cond)  {  // any label works  **goto** done;  }  // some code  …  **done** **// jumps to here**  // more code | | A not common jumping statement  - Jumps to line of code with label and continues from there | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **String** | | | | | | | | | | | | | | |
| - Methods that manipulate strings return new string objects  - Is **always** a reference type but behaves like a value type  - Immutable, method relating to string returns a **new** string | | | | | | | | | | | | | | |
| string str = “Hello” | | Instantiate a new string | | | | | | | | | | | |  |
| “str” + var | | String concatenation | | | | | | | | | | | | “Hello “ + args[0] |
| $”Hello {var}” | | String interpolation  - Place dollar sign before the start of the string and expression to **evaluate within curly braces**  - Alternative to string concatenation  - Avoids breaking up the string | | | | | | | | | | | | $”Hello {args[0]}” |
| str.Length | | Returns the number of characters in string | | | | | | | | | | | |  |
| .TrimStart() | | Remove leading spaces | | | | | | | | | | | |  |
| .TrimEnd() | | Remove trailing spaces | | | | | | | | | | | |  |
| .Trim() | | Remove both leading and trailing spaces | | | | | | | | | | | |  |
| .Replace(pattern, target) | | Replace target word with pattern in string | | | | | | | | | | | |  |
| .ToUpper() | | Convert to all caps | | | | | | | | | | | |  |
| .ToLower() | | Convert to all lowercase | | | | | | | | | | | |  |
| .Containers(keyword) | | Returns Boolean whether string contains keyword as a substring | | | | | | | | | | | |  |
| String.IsNullOrEmpty(str) | | Returns if a string is null or empty  - Class method | | | | | | | | | | | |  |
| Strings.Left(str, length) | | Returns a string containing a specified number of characters from the left side of a string  - Start is one-based  - Length cannot be 0 and less | | | | | | | | | | | |  |
| Strings.Mid(str, start, length)  Strings.Mid(str, start) | | Returns a string that contains all the characters from a specified position in a string  - Start index is inclusive  - If length not specified, returns substring from start index | | | | | | | | | | | | Example:  // Returns “vator”  String.Mid(“elevator”, 4, 5)  // Returns “day”  String.Mid(“terrible day”, 10) |
| Strings.Right(str, length) | | Returns a string containing a specified number of characters from the right of the string  - i.e. Substring from the right, count number of char | | | | | | | | | | | | Example:  // Returns “”duck”  String.Right(“psyduck”, 4) |
| **Array** | | | | | | | | | | | | | | |
| arr.length | | Length of array | | | | | | | | | | | |  |
| double[] arr = **new** double[int]; | | Initialise new array with specified size/length | | | | | | | | | | | | **Error:** “Unassigned local variable” since variable pointing to array that exists  **double[] numbers;**  numbers[0] = 12.7; |
| arr[0] = value | | Populate array | | | | | | | | | | | |  |
| **var** arr = **new[]** {1, 2, 3}; | | Initialise array with values  - No need to specify length  - Use curly brackets | | | | | | | | | | | |  |
| **Mathematical Operations** | | | | | | | | | | | | | | |
| +, -, \*, / | | Addition, subtraction, multiplication, division | | | | | | | | | | | | += |
| % | | Remainder operator/modulo | | | | | | | | | | | |  |
| Math.min(x, y) | | Gives the lower valued number between 2 values | | | | | | | | | | | |  |
| Numeric types | | Type | | | | |  | | | | | | | |
| int | | | | | Integers  - Note: Integer division always produces an integer result | | | | | | | |
| float | | | | | Single precision numbers  - Represent non-integral numbers that may be very large or small in magnitude | | | | | | | |
| double | | | | | Double precision numbers (commonly used)  - Greater range than integer values  - Has twice as many number of binary digits as single-precision | | | | | | | |
| decimal | | | | | - Smaller range but greater precision than double | | | | | | | |
| .MinValue  .MaxValue | | Smallest/highest possible value  - Constant value | | | | | | | | | | | | E.g.  int.MinValue |
| Underflow/Overflow | | Condition when value that exceeds minimum and maximum limits  - Value wrap to the other limit | | | | | | | | | | | |  |
| Math.PI | |  | | | | | | | | | | | |  |
| decimal c = 1.0M; | | Use M suffix on numbers to indicate that a constant should use the decimal type (otherwise compiler assumes double type) | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **(Formatting) Types** | | | | | | | | | | | | | | |
| $”{result:N3}” | | Format to keep 3 decimal places | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **List** | | | | | | | | | | | | | | |
| - Dynamically-sized array | | | | | | | | | | | | | | |
| Systems.Collections.Generic | | Namespace | | | | | | | | | | | |  |
| List<double> lst = **new List<**double**>** (); | | Initialise new list | | | | | | | | | | | |  |
| new List<double>() {value1, value2, …} | | Initialise new list with values | | | | | | | | | | | |  |
| lst.Add(value);  **OR**  lst[0] = value; | | Add new value to array | | | | | | | | | | | |  |
| lst.Remove(value); | |  | | | | | | | | | | | |  |
| **.Count** | | Number of elements in array  - A property of the data structure, not a method | | | | | | | | | | | |  |
| .indexOf(value); | | Returns index of item  - Returns -1 if item isn’t in list | | | | | | | | | | | |  |
| .Sort() | | Sorts all items in the list in their normal order  - Sorts alphabetically for strings | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **Data Types** | | | | | | | | | | | | | | |
| Metadata view of data type | | 1. Cursor over keyword of type  2. Press F12 | | | | | | | | | | | | If type is a struct, it is a reference type |
| Structure type  (for classes) | | - A special type (pass-by-value)  - Not commonly used  - Can encapsulate data and related functionality | | | | | | | | | | | | public **struct** Coords  {  public Coords(double x)  {  …  }  **public double X { get; }**  …  } |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **(User-defined) Classes and Objects** | | | | | | | | | | | | | | |
| Book book = new Book(); | | | Given a class, instantiate a new object from the class | | | | | | | | | | |  |
| [access modifier] class Meepit  {  **public** Meepit**()**  {  …  }  } | | | Custom constructor | | | | | | | | | | |  |
| // In method call  SetName(**ref** book1, …)  // In method definition  … SetName(**ref** Book b, …)  OR  // In method call  SetName(**out** book1, …)  // In method definition  … SetName(**out** Book b, …) | | | Pass by reference  - Gives the method a pointer referencing memory location of object  - Use either keyword **ref** or **out**  - Using keyword out  - C# compiler assumes that the incoming reference has not been initalised  - An error will occur if no **value is assigned to out parameter** in method definition | | | | | | | | | | |  |
| Access Modifiers | - For classes and methods | | | | | | | | | | | | | |
| Keyword | | | | |  | | | | | | | | |
| public | | | | | Always have uppercase name | | | | | | | | |
| private | | | | |  | | | | | | | | |
| internal | | | | | Class can only be used inside the same .csproj project (default if not defined) | | | | | | | | |
| private string name;  public string Name  {  // getter  **get**  {  return name;  }  // setter  **set**  {  name = value;  }  } | | Add **properties** to a variable, define its **setter** and **getter** method using get, set keywords  - Instead of defining separate, do it together in one block  - Note that there are no parentheses after the method name | | | | | | | | | | | | |
| **get** keyword | | | | | | | | | | **set** keyword | | |
|  | | | | | | | | | | - As if written a setter method with a parameter named value (implicit typing) | | |
| How to use:  book.Name | | | | | | | | | | How to use:  book.Name = “”; | | |
| public string Name  {  **get; set;**  }  **OR**  public string Name  {  get;  private set;  } | | Shorter alternative (auto-property)  - No longer need to explicitly declaring the field  - Can specify access modifier | | | | | | | | | |  | | |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **Exception Handling** | | | | | | | | | | | | | | |
| **try**  {  return array[index];  }  **catch** (Exception e)  {  **throw new** System.ArgumentOutOfRangeException(“message”, e);  }  **finally**  {  …  } | | | | | | | | | | Try-catch statement  - Catch and handle an exception generated by working code  - Make sure that the appropriate exception is thrown | | | | |
| Possible exceptions to catch | | - For within catch(…) keyword  - Can stack as many catch blocks  - Example:  …  catch (…) { … }  catch (…) { … }  … | | | | | | | | | | | | |
| Exceptions | | | | | | | | |  | | | |
| Exception e | | | | | | | | | Catch any type of exception | | | |
| System.IndexOutOfRangeException e | | | | | | | | |  | | | |
| e.Message | |  | | | | | | | | | | | |  |
| ArgumentException(“…”) | |  | | | | | | | | | | | |  |
| NotImplementedException(); | |  | | | | | | | | | | | |  |
| NullReferenceException() | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **Constants** | | | | | | | | | | | | | | |
| **readonly** string x;  readonly string x = “Hello”; | | readonly objects can only initialise, change or write to (in the contructor) | | | | | | | | | | | |  |
| **const** string GREETING = “Hi”; | | Can only instantiate once and remains unmodified after  - Can be publicly accessible, don’t need to write with property (i.e. get, set)  - Treated like static members of a class (reference by type name. not object) | | | | | | | | | | | |  |
| **Delegates** and **Events** | | | | | | | | | | | | | | |
| Delegate | | Pointer to a function  - Delegate is a variable that can point to and invoke different methods (of a specific shape and structure) | | | | | | | | | | | |  |
| public **delegate** string WriteLogDelegate(string logMessage);  [Fact]  public void WriteLogDelegateCanPointToMethod()  {  **// METHOD 1**  WriteLogDelegate log;  // pass in a function  log = new WriteLogDelegate(ReturnMessage);  var result = log(“Hello!”);  Assert.Equal(“Hello!”, result);  }  string ReturnMessage(string message)  {  return message;  } | | | | | | | | | - Most important thing to match is the return type  - Useful in cases like in software engineering where a *watcher* is needed, such that **when something is updated**, **a list of people**/multiple components are **notified** (but is it not the job of the updated object to keep track)  - Can only add or remove method from invocation list  - Cannot assign list to something (e.g. null) | | | | | |
| int count = 0;  [Fact]  public void WriteLogDelegateCanPointToMethod()  {  **// METHOD 2**  WriteLogDelegate log = ReturnMessage;  // invoke other functions as well  **// add methods to invocation list \*\***  log **+=** ReturnMessage;  log **+=** IncrementCount;  var result = log(“Hello!”);  Assert.Equal(“Hello!”, result);  }  string ReturnMessage(string message)  {  count++;  return message;  }  // Another method, matching return type to delegate  string IncrementalCount(string message)  {  count++;  return message.ToLower();  } | | | | | | | | |
| Events | | - Events often used in programming frameworks like Windows Presentation Foundation (WPF), Xamarin.Forms, Windows Forms, ASP.NET Web Forms (i.e. **forms, desktop programming)** | | | | | | | | | | | |  |
| public delegate void GradeAddedDelegate(  **object** sender, **EventArgs** args);  public **event** GradeAddedDelegate GradeAdded;  // method where update occurs, also call delegate  public void AddGrade(double grade)  {  …  GradeAdded(**this**, **new EventArgs());**  }  // In Program.cs  // In Main method  book.GradeAdded += onGradeAdded;  static void OnGradeAdded(object sender, EventArgs e)  {  …  } | | | | | | | | | Arguments | | | | Explanation | |
| object sender | | | | Object that is invoking the delegate | |
| EventArgs args | | | | Additional information about event | |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| **Object-Oriented Programming** | | | | | | | | | | | | | | |
| public class **ChildClass : ParentClass**  {  //constructor  public ChildClass(…) **: base(…)**  {  …  }  public **override** void AddGrade(double grade)  {  …  }  } | | | | | Define a child class  - Indicate parent class to inherit from after semi-colon (:)  - In constructor of child class, use keyword base to pass arguments into constructor of parent class  - Only **single** inheritance allowed  - Override method in the parent class | | | | | | | | | - All classes are derived from base class System.object or just object |
| public class ChildClass : AbstractClass, InterfaceClass  {  …  } | | | | | Inherit an abstract class and implement an interface  - Separate with comma (,) | | | | | | | | |  |
| Abstract class | | | | | | | | Interface | | | | | | |
| *inherits* | | | | | | | | *implements* | | | | | | |
| - May contain state and/or implementation | | | | | | | | - Can have no state or implementation | | | | | | |
| - Can be inherited without implementing the abstract method (although such a child class will be considered abstract as well) | | | | | | | | - A class that implements it must provide an implementation to **all** methods | | | | | | |
| - A class can only inherit from one abstract class | | | | | | | | - A class can implement multiple interfaces | | | | | | |
| public **abstract** class BookBase  {  public void AddGrade(double grade);  } | | | | | Define an abstract class  - Can also inherit from a class | | | | | | | | |  |
| public **interface** IBook  {  void AddGrade(double grade);  Statistics GetStatistics();  string Name { get; };  event GradeAddedDelegate GradeAdded;  } | | | | | Define an interface  - Usually named starting with the letter I | | | | | | | | |  |
| public **virtual** Statistics GetStatistics()  {  …  } | | | | | Defining a virtual method in a class  - Allows derived/child class to override implementation details for this method | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
| - | | | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |
|  | |  | | | | | | | | | | | |  |