Cheatography

C# / .NET Introduction Cheat Sheet

by VoltaicGRiD (VoltaicGRiD) via cheatography.com/164411/cs/34452/

Whats What

Class	A class can contain zero or more methods, zero or more constructors, and zero or more variables
Method	A method declares, simply, an action and, if necessary, response in the form of a 'return'. Methods can take variables as arguments to pass data along
Constructor	A method that auto-executes upon the creation of a new instance of a class
Return	The "output" variable of a method used to return data to the method's caller
Argument / Parameter	An "input" variable for a method used when calling said method to pass data to it

Good Beginner Projects

Chess Game - Create a game of chess with a UI (WPF / WinForms) and utilizing icons from FontAwesome that can be played between two players

Calculator - Create a calculator with a UI (WPF / WinForms) that can be used to perform basic mathematical operations on two numbers (addition, subtraction, multiplication, division, square root, exponents)

To-Do List - Create a simple UI (WPF / WinForms) where to-do items can be created, and marked off. Marked off to-do's should be retrievable after marked as complete

Weight Conversion Tool - A simple console or UI application where a weight type (lbs, kgs, oz) can be converted to any other weight type

Single-page Website - Utilize ASP.NET and the options presented when creating a new project to build a single-page website for yourself

Diagnosing Exceptions (A.K.A. Runtime Errors)

Error Message	Most Likely Cause
`NullReferenceException`	A variable or method is referencing a variable or method that doesn't exist or has been disposed of
`StackOverflowException`	Most often caused by too many consecutive method calls
`IndexOutOfRangeException`	Caused by an attempt to access (read/write) to an array object that doesn't yet exist, or is outside the min/max bounds of the array or list
`ArithmeticException`	A failure to cast, convert, or apply arithmetic (mathematical) operations on one or more variables
`NotFiniteNumberException`	When a floating-point (float) variable reaches positive or negative infinity
`IO.IOException`	A failure most often caused by an inability to read or write to a file or folder (either because of permissions, or a locked file)
`IO.FileNotFoundException`	Program is unable to find the file in question, and cannot perform any operation going forward

Cheatography

C# / .NET Introduction Cheat Sheet

by VoltaicGRiD (VoltaicGRiD) via cheatography.com/164411/cs/34452/

Diagnosing Exceptions (A.K.A. Runtime Errors) (cont)

`IO.DirectoryNotFoundException`	Same as above, except in reference to a folder or directory

Object Types

int - Integer	**double** - Floating binary-point number (exact)
float - Floating binary-point number (estimate)	**decimal** - Decimal number (exact)
string - Char array	**char** - Character
bool - True/false boolean	**var** - Dynamic variable

Using Statements

Using statements are declarations of which code libraries are being utilized within this file / class. Additionally, if declared similarly to a variable, a shortened keyword can be used in place of a full-length declaration further within the code.

For example:

```
using System;
using System.Di agn ostics;
using System.Wi ndo ws.F orms;
using Excel = Micros oft.Of -
fic e.I nte rop.Excel;
```

Using the Console

Comments

//	Single-line comment
/*	Begin multi-line comment
***/**	End multi-line comment

Declaring a Variable

Declaring a variable is as easy as knowing the type of variable needed, a name for the variable, and the default (a.k.a. initial) value for said variable.

```
{Variable Type} {Variable Name}
= {Default / Initial Value};
```

Say we need to create an integer named "-itemCount":

```
int itemCount = 0;
```

Now what about a true/false statement named "validated":

```
bool validated = false;
```

A variable based off a class named "Car" with a variable named "newCar":

```
Car newCar = new Car();
```

Declaring a Method

Declaring a method is nearly as simple as declaring a variable, you must know the publicity of the method, the return type (if one is needed), the name of the method, and any arguments (data) that is needed by the method. Remember, you create the name of the method.

```
{Publi city} {Return Type}
{Method Name} ({Argu ments})
```

Lets say we need a public method that returns an integer named "Add" that takes two integers as arguments:

```
public int Add (int number1, int
number2)
```

We can also declare a method with an optional parameter by declaring an argument with a default value:

Declaring a Method (cont)

```
public int Add (int number1, int
number2, int number3 = 0)
```

Alternatively, we can create methods that don't return anything:

```
public void DoSome thing ()
```

Publicity / Access Modifiers

public	Accessible from any class within any assembly or namespace
internal	Accessible from any class within only this assembly
protected	Accessible only by this class and any class derived or inherited by this class
private	Accessible only by this class

Access modifiers can be used on Methods, Classes, and even Variables, to allow for granular access to only the necessary aspects of your code.

Access Modifiers

Utilizing Git (Building a Portfolio) WIP

Firstly, ensure you have a GitHub or other Git application account. Most beginning / solo developers prefer GitHub for its ease-of-use and issue tracking capabilities.

Once an account is created, create a project (a.k.a. repository) and name it appropriately in the application, include as much detail, and a description, as possible.

From the newly created project's dashboard, look for a button to clone the repository to your machine. A few options may be presented, and if using both Visual Studio and GitHub, the 'Clone to Visual Studio' magnet button is a quick and efficient way of cloning the repository locally.

`var input = Console.ReadLine();`	Gets the last line input to the console and saves it to variable 'input'
`Console.Write({{x}});`	Writes the text-r-eprese-ntation of the variable (any) and does not move to the next line
`Console.WriteLine({{x}});`	Writes the text-r-eprese-ntation of the variable (any) and moves to the next line

Use a variable name in place of {{x}}

Logical Operators	
==	Equal to
!=	Not equal to
&&	And (i.e. A == 1 && B == 2)
||	Or (i.e. A == 1 || B == 2)
>=	Greater than or equal to
<=	Less than or equal to
>	Greater than
<	Less than

By **VoltaicGRiD** (VoltaicGRiD)
cheatography.com/voltaicgrid/

Not published yet.
Last updated 7th October, 2022.