Use var when assign the value at the same time of declaring the variable

Var test = 1;

User string or other keywords when you do not assign a value when declaring the variable.

Int test;

Value type always stored in the stack – quick access. Store the actual value

Object based type (reference type like array) is stored in the hip. Store the reference (pointer) in the RAM. When do comparison, It will check the pointer and hash value.

Boxing: turning value type to reference type or wise versa;

Control flow: if, loop, switch.

Is operator: check data type – is int, is null

Bool? Thing; - ? force it to be null

Reference type are nullable – array and object

Value types are non-nullable export for string. String can be null

System.nullreferenceexeception = can not read property of undefined

Var1 ?? var2, if var1 is not null, return var1. If var2 is not null, return var2

Void means not returning anything

static void Main(string[] args): method signature.

Field, property, method are three things you can have in a class

Field is kinda like state in react, available internally.

You can not write code directly inside of class like:

namespace NameParser.Names

{

class FirstName

{

Console.WriteLine("Enter your first name");

var firstName = Console.ReadLine();

foreach (var letter in firstName)

{

Console.WriteLine(letter);

}

}

}

Every class has default constructor

Field – property – constructor – method

#region Methods #endregion -- shortcut ctrl + k, s

Abstract class can not be instantiated. It can only be inherited . The method has to be overwritten

Override – overwrite the base method. Use it along with virtual keyword

Virtual – you can override the method, but not requried

When inherited, you can access all methods in base class using base.xxx() keyword

Use protected key to make method or property accessible for inherited class

Non-Primitive Types:

Class, struct, array, string,Enum

Declaring Classes

Public class Person {} – access modifier, class keyword, class name

When creating objects (instantiated class), you will need to allocate memory for the object

CLR has garbage collections that will remove unused objects to free up memory

Person person = new Person() is equivalent to var person = new Person()

Static modifier:

Public class Calculator

{

Public static int Add(int a, int b)

{

Return a + b;

}

}

Int result = Calculator.Add(1,2)

With static keyword, you can access the method directly without instantiate new object. In fact, you can not instantiate new object with class that has keyword static. Use static when you only want one single instance to exist. For instance, only one datetime needs to be existed in memory. Console.WriteLine() is another example that using static keyword.

Array:

Int[] numbers = new int[3] {1,2,3}

Or

Var numbers = new int[3]

First [] is to tell complier we want to create an array, the second [] is to define the size of the array. And {} is used to initialize the elements for the array.

In C#, array size can not be changed. When declare an array, you need to allocate memory for it. That is why you need new keyword.

String:

String format:

string.format(“{0} {1}”, firstName,lastName);

string is immutable, once is created, it can not be changed.

Verbatim strings: string path = @”c:\project\..”;

Enum:

A set of name/value pairs (constants)

Enum is like class, so need to be declared in the namespace level

Used Enum when you declare multiple related constants:

Bad:

Const int RegularAirMail = 1;

Const int RegisteredAirMail = 2;

Good:

Public enum ShippingMethod

{

RegularAirMail = 1,

RegisteredAirMail = 2,

Express = 3;

}

Var method = ShippingMethod.Express

Parsing: convert the string to different type like int

Value type vs reference type

Value type – structure – all primitive types – saved in the stack, memory gets allocated automatically

Reference type – class – all not primitive types – saved in heap, you need to allocate memory

Value type:

var a = 10;

var b = a;

b++

a = 10; b = 11

The value from a gets copied over to b, and b increment. B is now independent from a;

Reference type:

Var array1 = new int[3] {1,2,3}

Var array2 = array1;

Array2[0] = 0;

Now array1[0] will also equal to 0

When copy the variable, the memory address that points to the heap get copied. Both copies are pointing to the same address. Changes made in the heap will affect both variables.

Loop:

User break to jump out of the loop, use continue to jump to the next iteration