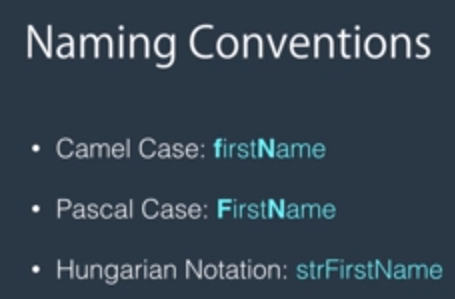
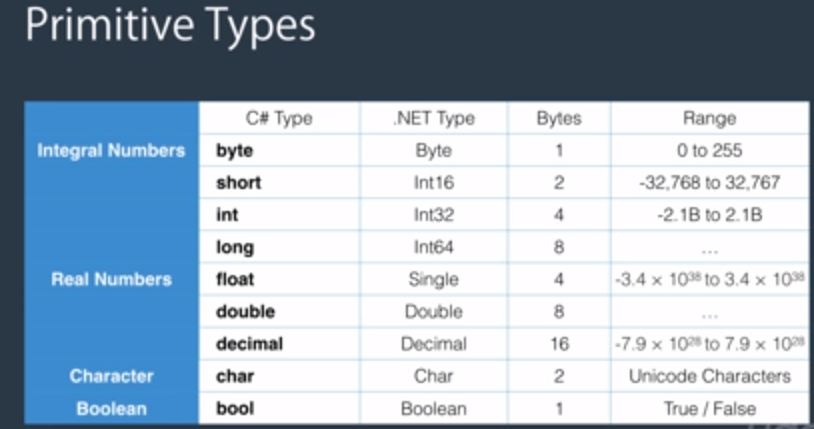
My Notes:

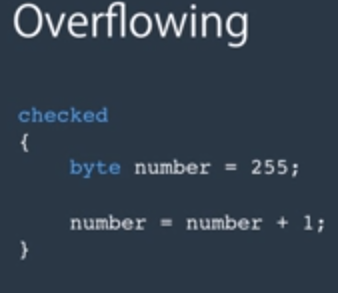
* @int can be used as variable name.
* Constant: const float Pi = 3.14f;
* Using const keyword or declaring an identifier as constant maintains a code security so that it cannot be changed accidently at run time.



* In Hungarian notation we used datatype in variable name.
* For variable naming use Camel Case, constant use Pascal case.

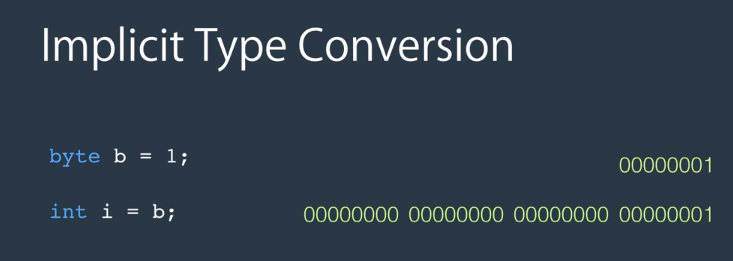


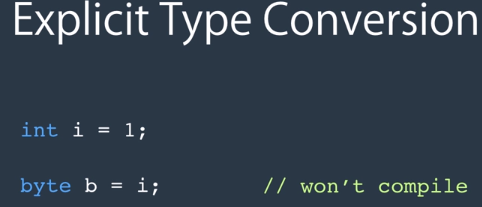
* decimal number = 1.2m [this is how we declare a decimal variable].



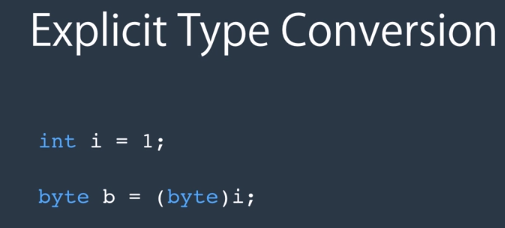
The above code will check if the variable value is overflowing, If that’s the case it will throw an exception.

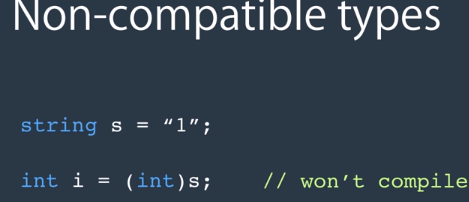
* Byte.MinValue, flaot.MaxValue shows the limit of the variable.

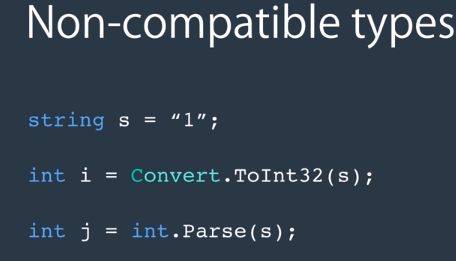


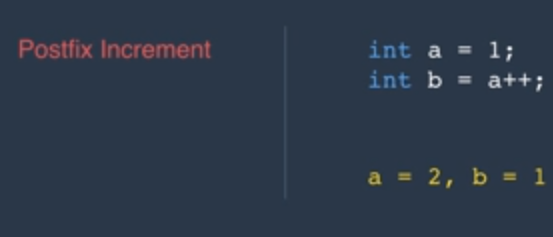


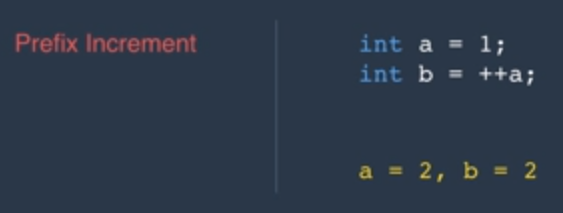
* Casting



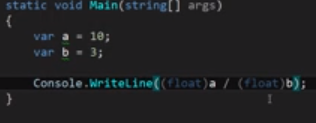
* 







* Division



The result for above calculation will be in floating point number i.e. 3.33333

But we don’t cat the variables then the result would be just an integer i.e. 3

* **“New”** operator in object declaration, all it is doing is allocating memory to the object.

We can also use var person = new Person();

* **Static Modifier:**

Use of static modifier is to make sure that only one instance of the class should be present in the memory, if we don’t use the static keyword then the object in memory is duplicated.

* Strcuts are useful when we are willing to create a light weight objects of same type, or even 1000 or 10’s of 1000 of objects of same type then structs are more efficient to use.
* Array: is a data structure to store a collection of variables of same type.

Int[] arrayName = new int[3]{1,2,5} – declared and defined at the same line.

We can also use var arrayName = new int[3];

New keyword confirm that Array is of type object or instance of class and is used to create a space in memory.

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

var formattedName = string.Join(“,”, names);

names is an array in above code.

O/P: John, Mary, Jack

* using @ before the path name in windows is called verbatim string.
* Enum by default are of type integer, if we want a different type then we have to do this way:

Public enum ShippingMethods: byte

{

RegularExpressMail = 1;

RegisteredmailbyAir = 2;

Express =3;

}

* We have to declare the Enum at namespace level that means outside of the class.
* Getting the name of the enum members suing the value of enum.

e.g. var methodId = 3

Console.Writeline((ShippingMethod)methodId);

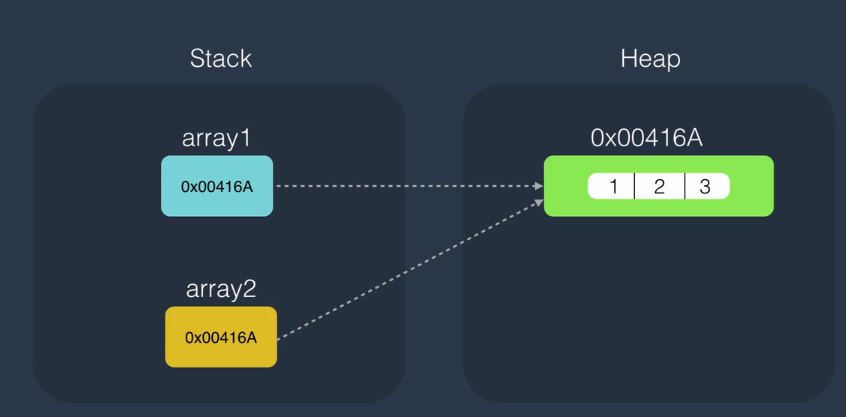
O/P: Express

* Parsing: getting a string and converting to other type.

e.g. parsing a string to type Shipping method:

var methodName = “Express”;

var shippingMethod = (ShippingMethods)Enum.Parse(typeof(ShippingMethods), methodName);

* All primitive types are structs.
* 
* 
* **Break**: To jump out of the loop, **Continue**: To continue the next iteration.
* Random Class:

Var randon = new Random();

for(var i=0, i <10, i++){

cw(random.Next());

cw(random.Next(1, 10));

}

* ASCII code: cw((int)’a’); -- prints ASCII code of the letter

Random characters:

Cw((char)(random.Next(97, 122))); -- prints characters (char casting) as we have characters ASCII code starts from 97 to 122.

Cw((char)(‘a’+random.Next(0,26))); -- Another way of generating random characters as we have added ‘a’ which represents a number and added to a number random.Next(0,26).

* New way to declare string

Var password = new string ()—Some overloads here

* Rectangular Array [2-D array]

Var array = new int[3(Rows),5(column)]

3-D Array:

Var array = new int[3, 5, 2]

* Jagged Array : At the top level this is collection of single dimension array so in below image there are 3 elements of single dimension array.

And each element is again another array of single dimension.

So the 1st element of jagged array is an array of size 4.

Declaration:

First declare top level array as:

Var array = new int[3][]; (3 is size of the top level array, 3 Rows)

Then array[0] = new int[4];

Array[1] = new int[5];

Array[2] = new int[3];

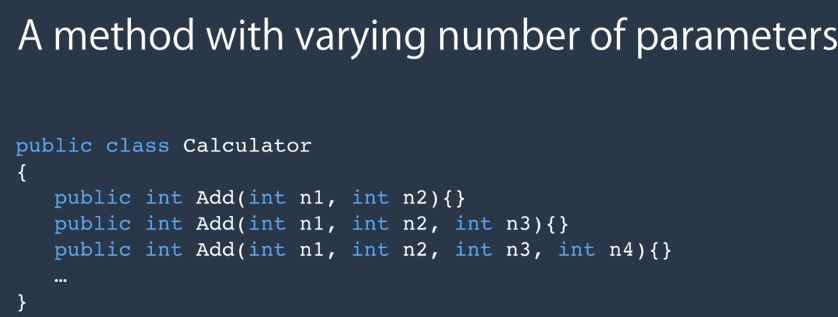


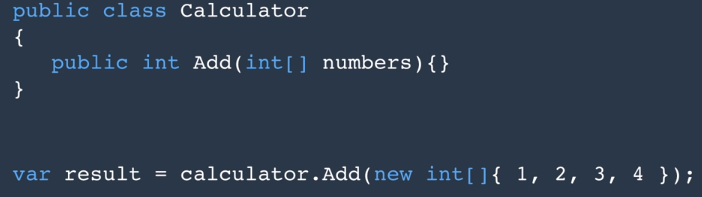
* Accessing above jagged array:

Array[0][0] = 1;

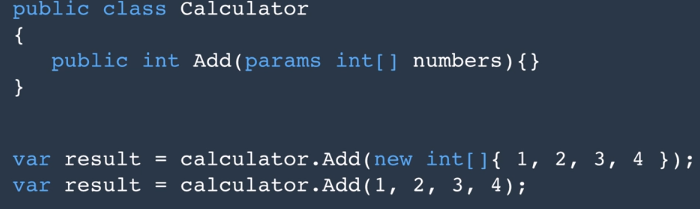
* File class always scans the path whether the user has the proper rights there or not.

Security checking is done only during instantiation of FileInfo, and that's why it's more efficient than File if you're executing multiple operations on a given file. Because security checking will be done only once.

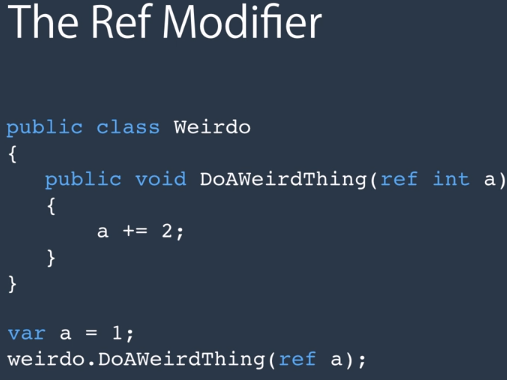


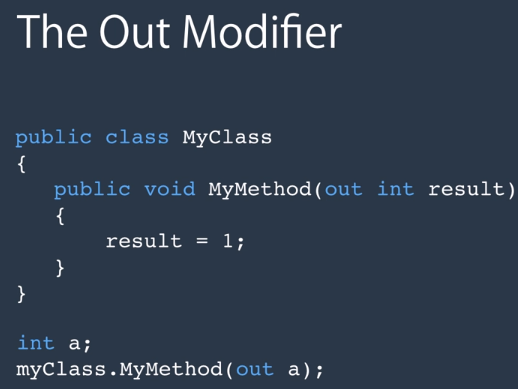


Use of param modifier for better solution: In this case we don’t have to initialize an array of integer we could directly use: calculator.Add(1,2,3,4); so we are avoiding the intialization of the array with new keyword.



* Ref and Out





* Try Parse Method: Doesn’t throw an exception

