As for what to use when:

* For values which are "naturally exact decimals" it's good to use decimal. This is usually suitable for any concepts invented by humans: financial values are the most obvious example, but there are others too. Consider the score given to divers or ice skaters, for example.
* For values which are more artefacts of nature which can't really be measured exactly anyway, float/double are more appropriate. For example, scientific data would usually be represented in this form. Here, the original values won't be "decimally accurate" to start with, so it's not important for the expected results to maintain the "decimal accuracy". Floating binary point types are much faster to work with than decimals.

**Local functions**

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/local-functions

**Static class**

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static>

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members>

**Local Functions**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/local-functions>

* Querying data from a database, IEnumerable execute a select query on the server side, load data in-memory on a client-side and then filter data.
* Querying data from a database, IQueryable execute the select query on the server side with all filters.
* IEnumerable exists in System.Collections Namespace.
* IQueryable exists in System. Linq Namespace.
* Both IEnumerable and IQueryable are forward collection.

|  |  |
| --- | --- |
| **Static Class** | **Non-Static Class** |
| Static class always contains static members. | Non-static class may contain both static and non-static methods. |
| Static class does not contain an instance constructor. | Non-static class contains an instance constructor. |

ECMA – <https://www.ecma-international.org/publications-and-standards/standards/ecma-334/>

-------------------------------------------------------------------------------------------------------------------------------

**Strongly type**

22

For e.g you cannot Multiply or Divide two different types i.e String vs Integer

var answer = 1 \* "1"; // you cannot do this

You have to explicity cast it, this is known as strongly typed

Where as if you see in php

$x = "3" \* 1; // is correct in php

So here you dont need to explicitly cast it.

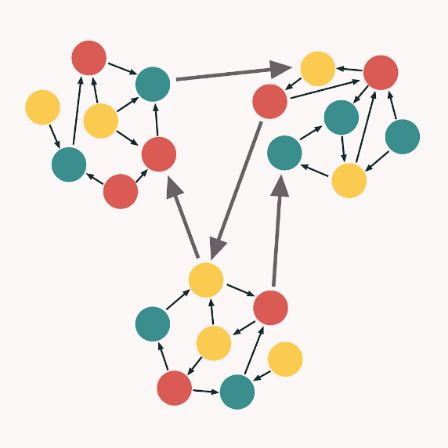
-----------------------------------------------------------------------------------------------------------

.net Framework

<https://en.wikipedia.org/wiki/.NET_Framework>

Binary files - A binary file is a computer file that is not a text file. The term "binary file" is often used as a term meaning "non-text file"

**high cohesion and low coupling**



Array

The array is the single most important data structure in computer programming. The computer itself is nothing but a collection of arrays of switches. A byte is an array of 8 binary bits. An integer (in many languages) is an array of 4 bytes and the order of the 4 bytes that make a 32 bit integer can vary even on a single machine (GPU’s can use different ordering than the CPU’s they talk to). A string is an array of characters (can be one or two bytes per character depending on encoding). All of a computer’s memory is just a big array of bytes (or words that are multiples of a byte).

When I first learned about all the advanced data structures like linked lists and binary trees, I assumed that they were what we were supposed to use and they would be fast and efficient. Unfortunately, they are not fast and efficient, and using them often complicates your code when you could have used a simple array. While it may seem harder to use ‘primitive’ structures like arrays, you will be rewarded by much faster performance and in many cases, simpler code. Arrays are also pretty consistent in coding syntax and behavior across all computer languages.

Arrays are fast because they let you directly access a single element’s data in one computer cycle regardless of the size of the array. Because of the operating system and virtual memory and all that, there may be a few more than one actual computer cycle involved to retrieve an element but the time to retrieve the first element is the same as the time to retrieve 100th element and the 500,000th element. No other data structure you build can be faster than an array. All the ‘advanced’ structures are implemented using arrays (and generally badly implemented because they need to be safe across all the different ways they can be used by programmers). If you need a custom data structure, build it on simple arrays and it will be fast.

**If you have set of values and the number of values will not change during the current process and you have an integer you can use or calculate easily to reference those values, use an array. You will be rewarded with simpler code that runs faster.**

**Value types vs reference types**

**Value type – Heap**

Arrow

Description automatically generated with medium confidence

**Reference ype**

Diagram

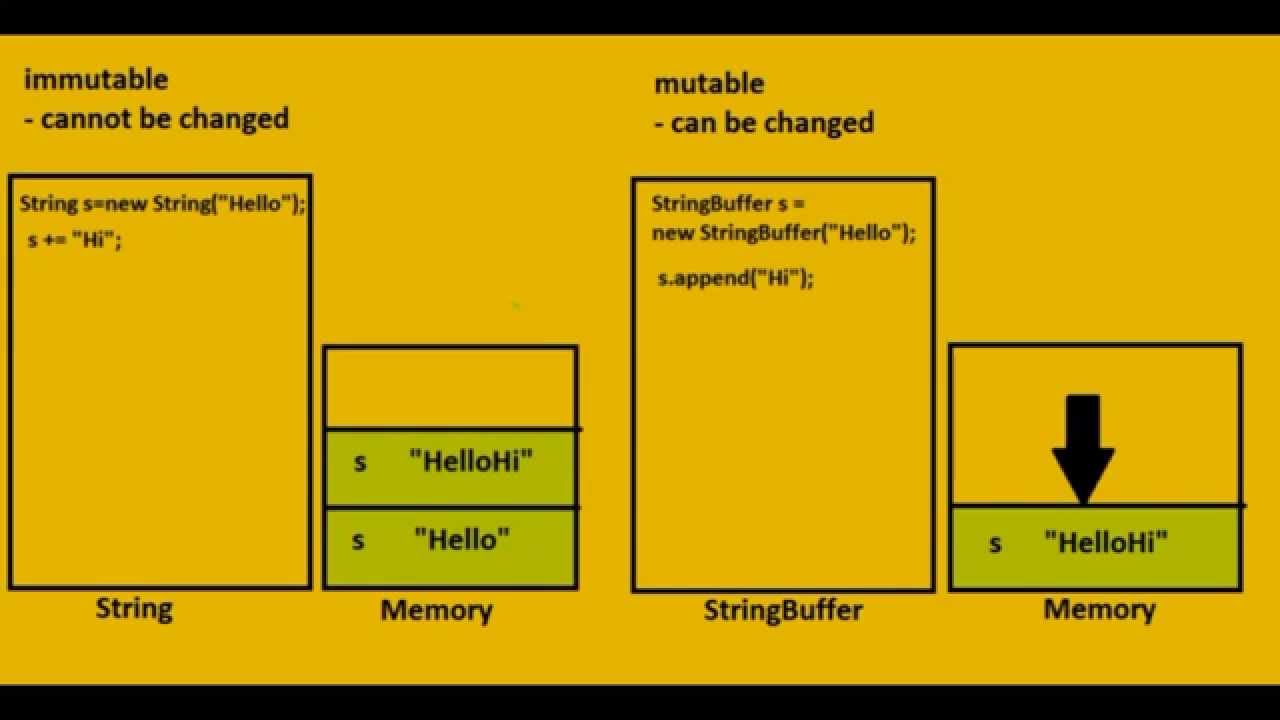
Description automatically generated

**Default value of Reference type = Null**

Diagram

Description automatically generated with medium confidence

**String vs String Builder**



StringBuilder is used to represent a *mutable string* of characters**. Mutable means the string which can be changed**. So String objects are immutable but StringBuilder is the mutable string type. It will not create a new modified instance of the current string object but do the modifications in the existing string object. The complete functionality of StringBuilder is provided by StringBuilder class which is present in System.Text namespace.

**Need of the StringBuilder:** As stated above that the String class objects are immutable which means that if the user modifies any string object it will result into the creation of a new string object. It makes the use of string costly. So when the user needs the repetitive operations on the string then the need of StringBuilder come into existence. It provides the optimized way to deal with the repetitive and multiple string manipulation operations.

|  |  |
| --- | --- |
| Method | Description |
| StringBuilder.Append | Appends information to the end of the current StringBuilder. |
| StringBuilder.AppendFormat | Replaces a format specifier passed in a string with formatted text. |
| StringBuilder.Insert | Inserts a string or object into the specified index of the current StringBuilder. |
| StringBuilder.Remove | Removes a specified number of characters from the current StringBuilder. |
| StringBuilder.Replace | Replaces a specified character at a specified index. |