C# Value and Reference Types

C# is a strongly and statically typed object-oriented programming language.

- Object-oriented is paradigm of thinking of how coding works. In an object-oriented programming language, we imagine things inside C# as objects rather than just simple code. Another paradigm is functional programming (most languages these days are object-oriented).

- Strongly means that once a variable's type is declared, it cannot be changed, however you can still change the value.

- Statically means each variable must have a type at compile time.

Stack, Heap, and Queue are three sections of our computer's memory.

- Stack: the memory held inside of the stack is stacked on top of each other, if you want to take something out you must take everything from on top off from it in order to reach data. (Think of pancakes). Using the stack will help reduce storage capacity.

- Heap: the memory held inside of the heap is stored in no order, this makes it easily accessible, and it also makes it where we can get rid of individual items inside of the heap very quickly. You can remove something that was just added, as easily as something that was added previously. However, most items stored in the heap will take more storage capacity.

- Queue: the memory held inside the queue is stored in a first in, first out fashion. The first item that goes into a queue is the first item that comes out. In the United States it is referred to as "a line".

A value type is set based on the provided value. Assigning one value type variable to other copies the contained value.

Examples include:

Boolean

- The boolean type is defined with the keyword ‘bool’. You can set the value to be ‘true’ or ‘false’.

Character

- The character type is defined with the keyword ‘char’. You can set the value to any single character surrounded by single quotes.

Decimal

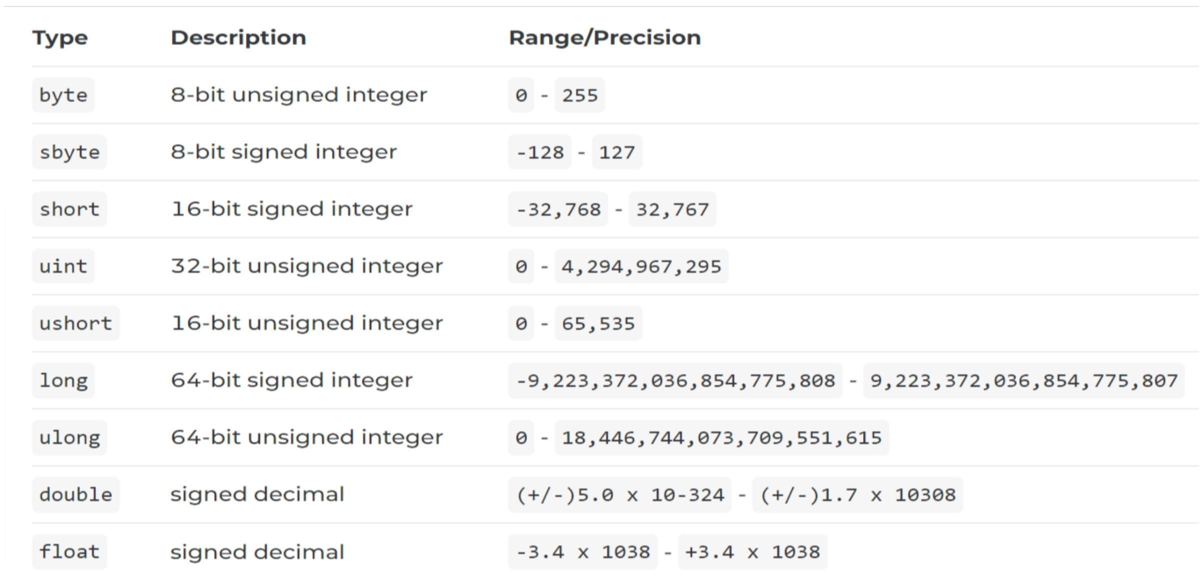
- The decimal type is defined with the keyword decimal. You can ser the value to be any decimal within the range of (-7.9 x 1028 to 7.9 x 1028) and ends with ‘m’.

Enum

- An Enum type is a distinct type used when you want to limit the value set to a fixed number of options.

Integer

- The integer type is defined with the keyword ‘int’. int is a 32-bit signed integer that can be set to any integer between -2,147,483,648 to 2,147,483,647.
    - 32 bit is the amount of RAM it occupies.
    - Signed means it can hold negative numbers. Unsigned means it can only be positive.



Type	Description	Range/Precision
`byte`	8-bit unsigned integer	`0` - `255`
`sbyte`	8-bit signed integer	`-128` - `127`
`short`	16-bit signed integer	`-32,768` - `32,767`
`uint`	32-bit unsigned integer	`0` - `4,294,967,295`
`ushort`	16-bit unsigned integer	`0` - `65,535`
`long`	64-bit signed integer	`-9,223,372,036,854,775,808` - `9,223,372,036,854,775,807`
`ulong`	64-bit unsigned integer	`0` - `18,446,744,073,709,551,615`
`double`	signed decimal	`(+/-)5.0 x 10-324` - `(+/-)1.7 x 10308`
`float`	signed decimal	`-3.4 x 1038` - `+3.4 x 1038`

The Keyword ‘null’ represents the absence of value.

- Reference types can be set to null, while value types require a non-null value. To make a reference type nullable, you can use the ‘?’ operator after the type declaration. For example, int? created a nullable integer type. Once a value type has been declared it can be assigned a null value by default.
- When you declare a nullable value type, the underlying type is still a value type, not a reference type. This means that nullable value types can still be stored on the stack rather than on the heap.
- Null is not the same as a 0 or an empty string. Null represents the absence of value: nothing is stored in memory or to disk.

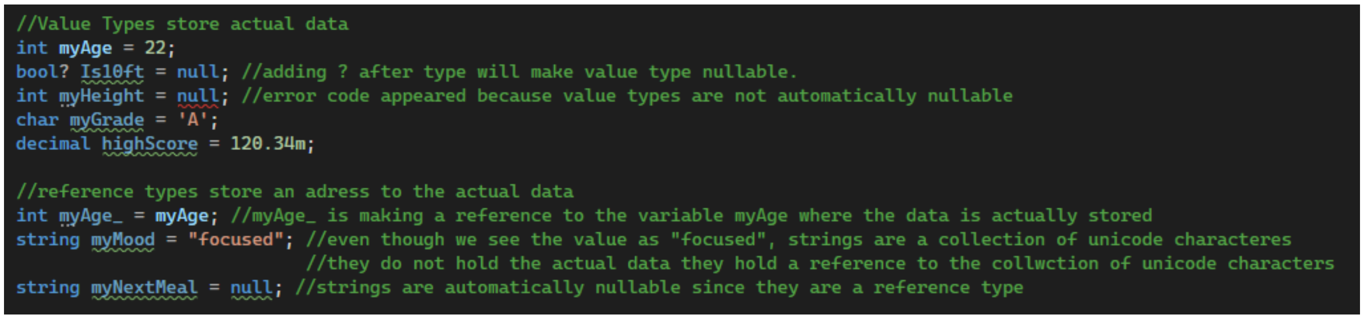
Reference Type vs Value Type

Value Types are stored in the stack. They store actual data rather than a reference to it. They are not automatically nullable.

Reference Types are stored in the heap. They are set by storing the actual date (object) in memory and storing a reference to the object within the variable. They automatically support being set to null. Reference types are objects that store references to the actual data. They act like a blueprint for example they wouldn’t store the value “mike” but they store how you ended up to “mike”.

Reference Types Examples:

- Classes
- Interfaces
- Objects
- Arrays: A collection of variables of the same type
    - Int[]myArray = new int[] {1,3,5,7,9};
- Strings

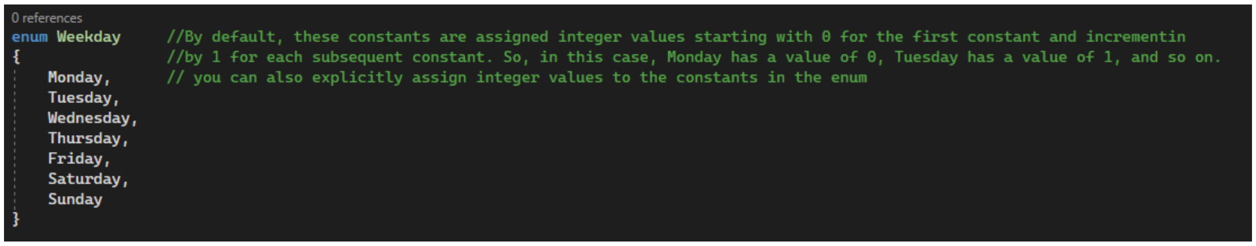


```
//Value Types store actual data
int myAge = 22;
bool? Is10ft = null; //adding ? after type will make value type nullable.
int myHeight = null; //error code appeared because value types are not automatically nullable
char myGrade = 'A';
decimal highScore = 120.34m;

//reference types store an adress to the actual data
int myAge_ = myAge; //myAge_ is making a reference to the variable myAge where the data is actually stored
string myMood = "focused"; //even though we see the value as "focused", strings are a collection of unicode characteres
                           //they do not hold the actual data they hold a reference to the collwction of unicode characters
string myNextMeal = null; //strings are automatically nullable since they are a reference type
```

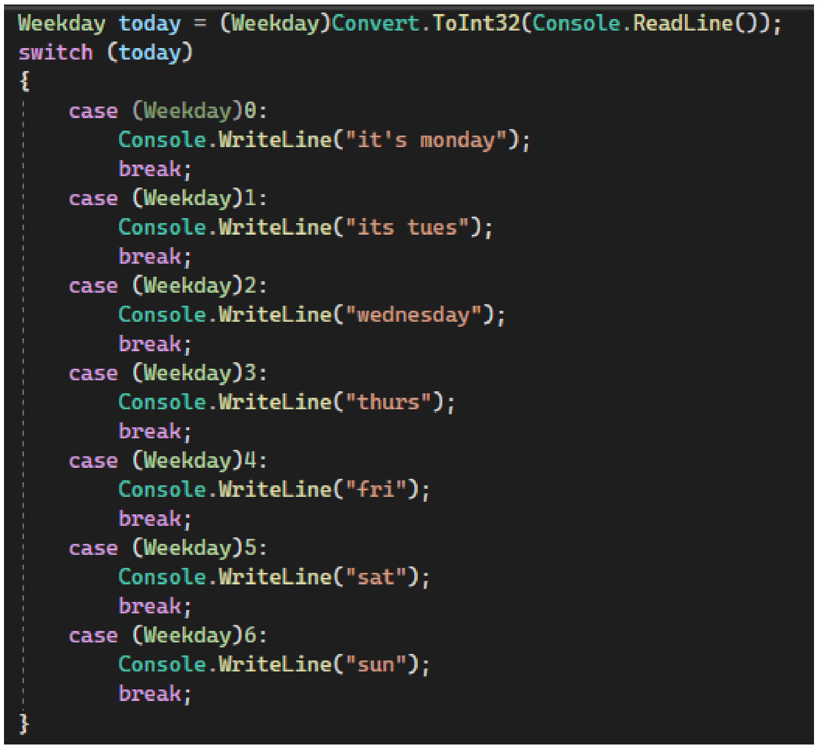
Enum is a value type that represents a set of named constant values.

Enums are often used to define a set of related, named constants that are assigned integer values



```
0 references
enum Weekday     //By default, these constants are assigned integer values starting with 0 for the first constant and incrementin
{                //by 1 for each subsequent constant. So, in this case, Monday has a value of 0, Tuesday has a value of 1, and so on.
    Monday,      // you can also explicitly assign integer values to the constants in the enum
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}
```

One useful feature of Enums is that you can use them in switch statements, like this:



```
Weekday today = (Weekday)Convert.ToInt32(Console.ReadLine());
switch (today)
{
    case (Weekday)0:
        Console.WriteLine("it's monday");
        break;
    case (Weekday)1:
        Console.WriteLine("its tues");
        break;
    case (Weekday)2:
        Console.WriteLine("wednesday");
        break;
    case (Weekday)3:
        Console.WriteLine("thurs");
        break;
    case (Weekday)4:
        Console.WriteLine("fri");
        break;
    case (Weekday)5:
        Console.WriteLine("sat");
        break;
    case (Weekday)6:
        Console.WriteLine("sun");
        break;
}
```

We already declared an Enum called Weekday and a variable called today of type weekday. We then use a switch statement to perform different cases based on the integer that represents the int typed.