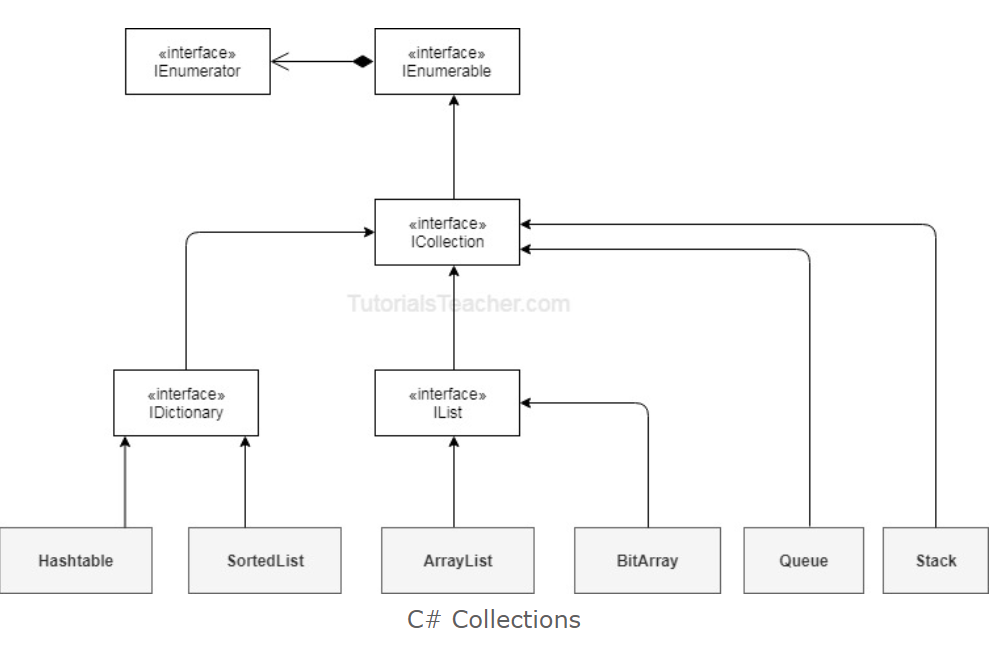
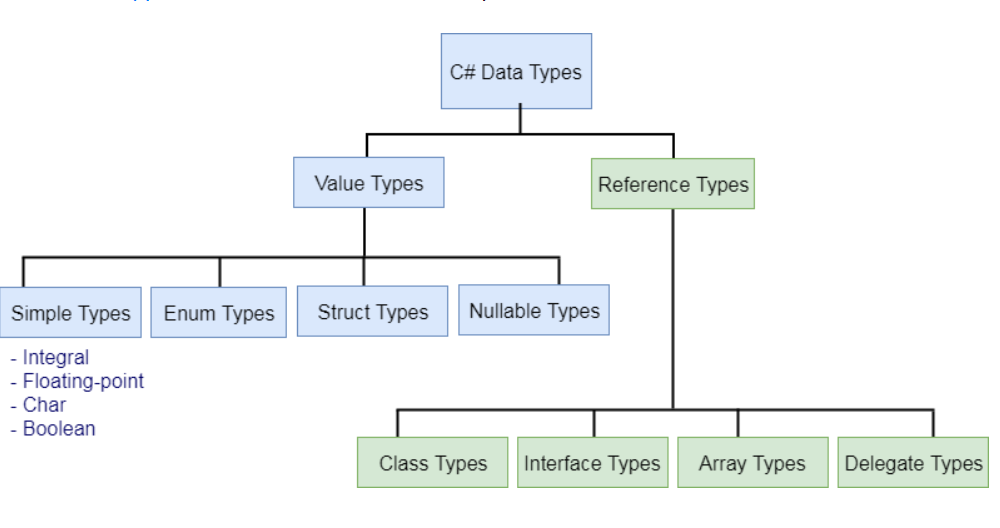
1. **C#:**
   * + - 1. Generics – some keyword to mention – compile time assign type.
         2. Collections:



* + 1. Struct
    2. Class - blue print of object
    3. Datatypes –



* + - 1. Value types – in Heap

**Value Type:**

A Value Type stores its contents in memory allocated on the stack. When you created a Value Type, a single space in memory is allocated to store the value and that variable directly holds a value. If you assign it to another variable, the value is copied directly and both variables work independently. Predefined datatypes, structures, enums are also value types, and work in the same way. Value types can be created at compile time and Stored in stack memory, because of this, Garbage collector can't access the stack.

* + - 1. Reference types – in Stack

**Reference Type:**

Reference Types are used by a reference which holds a reference (address) to the object but not the object itself. Because reference types represent the address of the variable rather than the data itself, assigning a reference variable to another doesn't copy the data. Instead it creates a second copy of the reference, which refers to the same location of the heap as the original value. Reference Type variables are stored in a different area of memory called the heap. This means that when a reference type variable is no longer used, it can be marked for garbage collection. Examples of reference types are Classes, Objects, Arrays, Indexers, Interfaces etc.

* + - 1. Var – anonymous type - implicitly typed variable – type based on right side of =
      2. Dynamic type - A dynamic type escapes type checking at compile time; instead, it resolves type at run time.
      3. Enum – named integer constants
      4. LINQ - language integrated query - single querying interface for different types of data sources.

var teenStudentsName = from s in studentList

where s.age > 12 && s.age < 20

select new { StudentName = s.StudentName };

* + - 1. Lambda – lambda expression along with linq – shorter way to represent anonymous method – special syntax

var studentNames = studentList.Where(s => s.Age > 18)

.Select(s => s)

.Where(st => st.StandardID > 0)

.Select(s => s.StudentName);

* + - 1. Extension method - additional methods. Extension methods allow you to inject additional methods without modifying, deriving or recompiling the original class, struct or interface

1. **public** **static** **class** XX
2. {
3. **public** **static** **void** NewMethod(**this** Class1 ob)
4. {
5. Console.WriteLine("Hello I m extended method");
6. }
7. }
   * + 1. Async - Async and await are the code markers, which marks code positions from where the control should resume after a task completes.
       2. pass by ref - memory based – same value
       3. pass by value – value will be different diff ram location address
       4. Optional parameters -

### **Comparison Chart**

| **BASIS FOR COMPARISON** | **BOXING** | **UNBOXING** |
| --- | --- | --- |
| Basic | Object type refers to the value type. | process of retrieving value from the boxed object. |
| Storage | The value stored on the stack is copied to the object stored on heap memory. | The object's value stored on the heap memory is copied to the value type stored on stack. |
| Conversion | Implicit conversion. | Explicit conversion. |
| Example | int n = 24; object ob = n; | int m = (int) ob; |

1. **Oops:**
   * 1. polymorphism

overloading - same method name attributes different – parameters

Overriding – same method name and attributes – inheritance – parent class method with virtual keyword – child class override keyword.

* 1. Encapsulation
  2. Interface – only method defn no content – used for multiple inheritance – not available in c#
  3. Abstract – method defn or full method – inheriting can override or implement

– eg>: cellphone old model features used in new ones.

* 1. Sealed Class – cannot inherit class – extension method is possible