1. Arry- Same type data store. It can’t accept null. It belongs to namespace system. array has a fixed size. An array is strongly-typed.  It means if we declare an Array of string type, then we cannot store the integer value in that array. The array provides better performance than the ArrayList because an array stores the same type of data which doesn't need unnecessary boxing or unboxing.
2. "Array class" is the base class for all arrays in C#. It is defined in system namespace

 An Array is reference type so memory for the array is allocated on the heap. We can initialize an Array using the "new" operator and by specifying the type and number of elements inside the Array.

ArrayList implements the IList interface. ArrayList is one of the most flexible data structures from C# collection. Collection classes are special classes for data storage and retrieval.

**EXAMPLE**

1. using System.Collection;
2. ArrayList a1 = **new** ArryList();
3. a1.add(**null**);
4. a1.insert(1, ”hi”);
5. a1.add(3);
6. a1.add(8.23);
7. ArrayLists do not have a specific size. When we initialize an arraylist, it will initially allocate the memory for 4 elements. When we add the 5th element, ArrayList will automatically redimension to double of its current size. So, the size will increase as 4, 8, 16, 32, 64 and so on (i.e 2^n).
8. ArrayList is a non-generic type of collection in C#. It means you can store any type of data in ArrayList.
9. ArrayList provides the facility of dynamic size but it comes at a cost of performance. The ArrayList's internal Array is of "object type". So, if we are using value type then each element is boxed and stored on a heap and whenever we access them it is unboxed to value type.
10. ArrayList implements the IList interface so, it provides a various method that we can use for easy implementation.

|  |  |  |
| --- | --- | --- |
| S.N | Struct | Classes |
| 1 | Structs are value types, allocated either on the stack or inline in containing types. | Classes are reference types, allocated on the heap and garbage-collected. |
| 2 | Allocations and de-allocations of value types are in general cheaper than allocations and de-allocations of reference types. | Assignments of large reference types are cheaper than assignments of large value types. |
| 3 | In structs, each variable contains its own copy of the data (except in the case of the ref and out parameter variables), and an operation on one variable does not affect another variable. | In classes, two variables can contain the reference of the same object and any operation on one variable can affect another variable. |

 In this way, struct should be used only when you are sure that,

* It logically represents a single value, like primitive types (int, double, etc.).
* It is immutable.
* It should not be boxed and un-boxed frequently.

In all other cases, you should define your types as classes.

A partial class is a special feature of C#. It provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled. A partial class is created by using a ***partial***keyword. This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.

What happens if the inherited interfaces have conflicting method names?

If we implement multiple interfaces in the same class with conflict method names, **we don't need to define all**. In other words, we can say if we have conflict methods in the same class, we can't implement their body independently in the same class because of the same name and same signature.

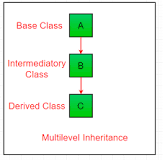
Can you implement multiple interfaces and if they have conflicting method names?

You **can implement one interface Explicitly and another implecitely**. ITest. Test will be the default implementation. So when calling from different class you will have to type cast the object into required Interface or Abstract class.

Can you inherit multiple interfaces?

**No you cannot inherit multiple interfaces**, because interfaces cannot be inherited. Interfaces are IMPLEMENTED, not inherited.

What is difference between inheritance and interface?

[[](https://www.google.com/search?rlz=1C1CHZN_enIN934IN934&output=search&tbm=isch&q=What+is+difference+between+inheritance+and+interface?&source=iu&ictx=1&fir=N-IuvD7we15d8M%252CAW8B2Cm1cghaCM%252C_&vet=1&usg=AI4_-kQyIg9keIKUAtDP_2UkQwsPZJC2FA&sa=X&ved=2ahUKEwjMr73dsYbzAhWl4zgGHfrwAQkQ9QF6BAgKEAE#imgrc=N-IuvD7we15d8M)](https://www.google.com/search?rlz=1C1CHZN_enIN934IN934&output=search&tbm=isch&q=What+is+difference+between+inheritance+and+interface?&source=iu&ictx=1&fir=N-IuvD7we15d8M%252CAW8B2Cm1cghaCM%252C_&vet=1&usg=AI4_-kQyIg9keIKUAtDP_2UkQwsPZJC2FA&sa=X&ved=2ahUKEwjMr73dsYbzAhWl4zgGHfrwAQkQ9QF6BAgKEAE" \l "imgrc=N-IuvD7we15d8M)

Inheritance is the mechanism by which one class is allowed to inherit the features of another class. Interface is the **blueprint** of the class. ... Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).