A namespace can contain types such as classes, structs, interfaces, enumerations, and delegates, in addition to other namespaces.

# Array:

In C#, arrays are actually objects, and not just addressable regions of contiguous memory as in C and C++.

int[] array1 = new int[] { 1, 3, 5, 7, 9 };

int[] array3;

array3 = new int[] { 1, 3, 5, 7, 9 }; // OK

//array3 = {1, 3, 5, 7, 9}; // Error

SomeType[] array4 = new SomeType[10];

The result of this statement depends on whether SomeType is a value type or a reference type. If it is a value type, the statement creates an array of 10 elements, each of which has the type SomeType. If SomeType is a reference type, the statement creates an array of 10 elements, each of which is initialized to a null reference.

int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };

int[][] jaggedArray = new int[3][];

Arrays can be passed as arguments to method parameters. Because arrays are reference types, the method **can change the value of the elements**.

var a = new[] { 1, 10, 100, 1000 }; // int[]

var b = new[] { "hello", null, "world" }; // string[]

C# is a strong Object Oriented programming language that is mostly built on the .NET framework.

C# is the airplane and .NET is the runway ;)

# Class and Struct

The variable remains in memory until all references to it go out of scope. At that time, the CLR marks it as eligible for garbage collection.

Because classes are reference types, a variable of a class object holds a reference to the address of the object on the managed heap. If a second object of the same type is assigned to the first object, then both variables refer to the object at that address.

public string Name { get; set; }

Because structs are value types, a variable of a struct object holds a copy of the entire object. Instances of structs can also be created by using the **new** operator, but this is not required

 This is one reason why structs are copied on assignment. By contrast, the memory that is allocated for a class instance is automatically reclaimed (garbage collected) by the common language runtime when all references to the object have gone out of scope. It is not possible to deterministically destroy a class object like you can in C++.

## Struct

* Within a struct declaration, fields cannot be initialized unless they are declared as const or static.
* A struct cannot declare a default constructor (a constructor without parameters) or a destructor.
* Structs are copied on assignment. When a struct is assigned to a new variable, all the data is copied, and any modification to the new copy does not change the data for the original copy. This is important to remember when working with collections of value types such as Dictionary<string, myStruct>.
* Structs are value types and classes are reference types.
* Unlike classes, structs can be instantiated without using a **new** operator.
* Structs can declare constructors that have parameters.
* A struct cannot inherit from another struct or class, and it cannot be the base of a class. All structs inherit directly from**System.ValueType**, which inherits from **System.Object**.
* A struct can implement interfaces.
* A struct can be used as a nullable type and can be assigned a null value.

Inheritance, together with encapsulation and polymorphism, is one of the three primary characteristics (or pillars) of object-oriented programming.

When a base class declares a method as [virtual](https://msdn.microsoft.com/en-us/library/9fkccyh4.aspx), a derived class can [override](https://msdn.microsoft.com/en-us/library/ebca9ah3.aspx) the method with its own implementation. If a base class declares a member as [abstract](https://msdn.microsoft.com/en-us/library/sf985hc5.aspx), that method must be overridden in any non-abstract class that directly inherits from that class. If a derived class is itself abstract, it inherits abstract members without implementing them. Abstract and virtual members are the basis for polymorphism, which is the second primary characteristic of object-oriented programming.

## Polymorphism

Polymorphism is often referred to as the third pillar of object-oriented programming, after encapsulation and inheritance.

## Dictionary

**Dictionary** 需要一个相等实现来确定键是否相等。可以使用一个接受 *comparer* 参数的构造函数来指定 [IEqualityComparer](https://msdn.microsoft.com/zh-cn/library/ms132151(v=vs.80).aspx) 泛型接口的实现；如果不指定实现，则使用默认的泛型相等比较器 [EqualityComparer.Default](https://msdn.microsoft.com/zh-cn/library/ms224763(v=vs.80).aspx)。如果类型 *TKey* 实现 [System.IEquatable](https://msdn.microsoft.com/zh-cn/library/ms131187(v=vs.80).aspx)泛型接口，则默认相等比较器会使用该实现。

Passing arguments by out and ref:

out: must be initialized by calee

ret: must be initialized by caller

# static, const, readonly

static member is accessed by class name, and does not belong any instance of class.

a filed cannot be declared as static const, a const field acts like static: be accessed by class name, must be initialized at declaration, the value of const member will be decided at compile time and cannot change afterwards.

readonly can be initialized at declaration(compile-time) or in constructor(run-time)

# Delegate

// Instantiate Del by using an anonymous method.

Del del3 = delegate(string name)

{ Console.WriteLine("Notification received for: {0}", name); };

// Instantiate Del by using a lambda expression.

Del del4 = name => { Console.WriteLine("Notification received for: {0}", name); };

# Using

1\ to import a namespace so that you can use types defined in the namespace

2\ using static System.Math.

so you can directly access static member in Math without class name.

3\create an alias for a namespace or type, like typedef

# Questions

**1\ the difference Abstract Class VS Interface**

The type is different, one is class , another is interface(no constructor).

* A class can implement any number of interfaces but a subclass can at most use only one abstract class.
* An abstract class can have non-abstract methods (concrete methods) while in case of interface all the methods has to be abstract.
* In an abstract class all data member or functions are private by default while in interface all are public, we can’t change them manually.
* In an abstract class we need to use abstract keyword to declare abstract methods while in an interface we don’t need to use that.
* An abstract class use constructor while in an interface we don’t have any type of constructor.

**2\ What is extension method in c# and how to use them?**

Extension methods enable you to add methods to existing types without creating a new derived type

**3\ string and StringBuilder**

String is an immutable object. Any change to the original string will lead to create a new object.

StringBuilder is mutable object. It’s helpful if you want to modify string frequently.

**4\ Partial class**

A partial class is only use to splits the definition of a class in two or more classes in a same source code file or more than one source files. Different parts should be in the same namespace.

**5\Difference between Equality Operator (==) and Equals() Method in C#.**

**6\nullable<>**

int? a = null;

**7\ object pool**

Object Pool is nothing but a container of objects that are ready for use. Whenever there is a request for a new object, the pool manager will take the request and it will be served by allocating an object from the pool.

Shortcomes:

Exception Handler

LINQ

Event Handler

String: a = b + c

// allocate new memory space, copy b and c to that memory, and variable a refers to that memory space.

StringBuilder a.append(b).append(c)

// operate in original space,