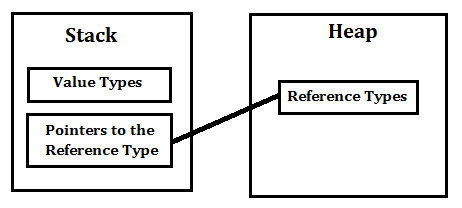
What is the difference between value and reference types? Is it true that the value type is always stored in a stack?

A Value Type holds the data within its own memory allocation and a Reference Type contains a pointer to another memory location that holds the real data. Reference Type variables are stored in the heap while Value Type variables are stored in the stack.



**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.

**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.

e.g.

In the above code the space required for the 20 integers that make up the array is allocated on the heap.

3. What is the difference between classes and structures? Is it possible to inherit from structure? In what cases it's better to use structures?

Technically speaking, structs and classes are almost equivalent, still there are many differences. The major difference like class provides the flexibility of combining data and methods (functions ) and it provides the re-usability called inheritance. Struct should typically be used for grouping data. The technical difference comes down to subtle issues about default visibility of members. Here you can see some of the Difference between Class and Structure.

Class is pass-by-reference and Struct is pass-by-copy, it means that, Class is a reference type and its object is created on the heap memory where as structure is a value type and its object is created on the stack memory

Class can create a subclass that will inherit parent's properties and methods, whereas Structure does not support the inheritance.

A class has all members private by default. A struct is a class where members are public by default.

Classes allow to perform cleanup (garbage collector) before object is deallocated because garbage collector works on heap memory. Objects are usually deallocated when instance is no longer referenced by other code. Structures can not be garbage collector so no efficient memory management.

Sizeof empty class is 1 Byte where as Sizeof empty structure is 0 Bytes

Classes are still fit for larger or complex objects and Structs are good for small, isolated model objects. Boxing and unboxing operations are used to convert between a struct type and object. Too much boxing and unboxing can have a negative impact on the heap, the garbage collector, and ultimately the performance of the application.

In general, it's perfectly possible to create structs that look a lot like classes and classes that look a lot like structs.

Which style you use depends on circumstances and taste. I usually prefer to use struct for classes that have all data public. I think of such classes as "not quite proper types, just data structures."

What is the purpose of namespaces? Is it good practice to keep your application in a single namespace?

Namespaces are used to organize the classes. It helps to control the scope of methods and classes in larger [**.Net**](https://www.geeksforgeeks.org/c-net-framework-basic-architecture-component-stack/)programming projects. In simpler words you can say that it provides a way to keep one set of names(like class names) different from other sets of names. The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace. It is also referred as **named group of classes** having common features. The members of a namespace can be **namespaces,**[**interfaces**](https://www.geeksforgeeks.org/c-interface/)**,**[**structures**](https://www.geeksforgeeks.org/c-structures-set-1/)**, and**[**delegates**](https://www.geeksforgeeks.org/c-delegates/)**.**

Can you describe why the **lock()** statement is designed to only accept reference type parameters?

When [a value type] object is passed from one thread to the other, a copy is made, so the threads end up working on 2 different objects, which is safe.

How method arguments are passed in C#? Can this behavior be changed?

В C # аргументы могут передаваться параметрам либо по значению, либо по ссылке. Передача по ссылке позволяет членам функции, методам, свойствам, индексаторам, операторам и конструкторам изменять значение параметров и сохранять это изменение в вызывающей среде. Чтобы передать параметр по ссылке с целью изменения значения, используйте ключевое слово ref или out. Для передачи по ссылке с целью избежать копирования, но не изменять значение, используйте модификатор in.

What is the difference between Int.Parse and Int.TryParse?

int.Parse() is a method to convert the given input into integer. Let's have an example to see this unique functionality.  
**Syntax**  
int.Parse(string s)

int.TryParse(input,out) is a method to convert the given input into integer, and the tryparse method is always used with out parameter to display default value. Let's have an example to see its unique functionality.   
**Syntax**  
int.TryParse(string s, out int result);

What are the implicit and explicit type conversions?

**Type Casting :**Conversion of one data type to another data type. and it can be done in two ways.

1. **Implicit Type Casting**
2. **Explicit Type Casting**

**Implicit type casting** is performed by the compiler on its own when it encounters a mixed data type expression in the program. it is also known as automatic conversion as it is done by compiler without programmer’s assistance. implicit casting doesn’t require a casting operator.

**Example :-**

1. **int** a=42;
2. **float** b=a;

here **b** will contain typecast value of **a**, because while assigning value to **b** compiler typecasts the value of **a** into float then assigns it to **b**.

**Explicit type casting** is performed by the programmer. In this type casting programmer tells compiler to type cast one data type to another data type using type casting operator. but there is some risk of information loss is there, so one needs to be careful while doing it.

**Example :-**

1. **float** a=42.12;
2. **int** b=(**int**)a;

here we explicitly converted float value of **a** to int while assigning it to int **b**. **(int)**is the type casting operator with the type in which you wants to convert.

How do you cast from one reference type to another without risking to throw an exception?

is operator

The is operator in C# is used to check the object type and it returns a bool value: **true** if the object is the same type and **false** if not.

For null objects, it returns false.

Syntax:

bool isobject = (Object is Type);

as operator

The as operator does the same job of is operator but the difference is instead of bool, it returns the **object** if they are compatible to that type, else it returns null.

Syntax:

Type obj = Object as Type;

Why isn't it possible to create an instance of an abstract class?

No, you cannot create an instance of an abstract class because it does not have a complete implementation. The purpose of an abstract class is to function as a base for subclasses. It acts like a template, or an empty or partially empty structure, you should extend it and build on it before you can use it.

An *abstract* type is *defined* largely as one that can't be created. You can create *subtypes* of it, but not of that type itself. The CLI will not let you do this.

An abstract class has a protected constructor (by default) allowing derived types to initialize it.

For example, the base-type Stream is abstract. Without a derived type **where would the data go**? What would *happen* when you call an abstract method? There would be no actual implementation of the method to invoke.

Is it possible to invoke a method from an abstract class?

Да любий не abstract method ;

Is it true that Interface can only contain method declarations?

An interface can be a member of a namespace or a class and can contain signatures of the following members:

Methods

Properties

Indexers

Events

Properties can be declared on interfaces. The declaration takes the following form: The accessor of an interface property does not have a body.

Thus, the purpose of the accessors is to indicate whether the property is read-write, read-only, or write-only.

Example:

// Interface Properties

interface IEmployee

{

string Name

{

get;

set;

}

int Counter

{

get;

}

}

Is it possible to specify access modifiers inside of an interface?

**Interfaces declared directly within a namespace can be declared as public or internal and, just like classes and structs, interfaces default to internal access.** Interface members are always public because the purpose of an interface is to enable other types to access a class or struct. No access modifiers can be applied to interface members.

Can we use Access Modifier with Interface

Yes, they can be declared as public or internal

By default Are inteface internal

what about Access Modfiers with Inteface members.

They are public. No access modifiers can be applied to interface members.

17. Can you inherit from two interfaces with the same method name in both of them?

Да, цей метод должен буть паблик при реализации

**public** void mymethod()

    {

        Console.WriteLine("GeeksforGeeks");

    }

АБО

void G1.mymethod()

{

    Console.WriteLine("GeeksforGeeks");

}

    // Here mymethod belongs to

    // G2 interface

    void G2.mymethod()

{

    Console.WriteLine("GeeksforGeeks");

}

Is it possible to define two methods with the same name and arguments, but with a different return types?

No, Console:Type 'Program' already defines a member called 'artem' with the same parameter types

What is the difference between method overriding and overloading?

**Difference between method overloading and method overriding**

|  |  |
| --- | --- |
| **Overloading** | **Overriding** |
| Having same method name with different Signatures (parameters). | Methods name and signatures must be same. |
| Overloading is the concept of compile time polymorphism. | Overriding is the concept of run time polymorphism. |
| Two functions having same name and return type, but with different type and/or number of arguments is called as Overloading. | When a function of base class is re-defined in the derived class called as Overriding. |
| It doesn't need inheritance. | It needs inheritance. |
| Method can be different access modifiers. | Method should be public. |

What does **protected internal** access modifier mean?

The protected internal keyword combination is a member access modifier. A protected internal member is accessible from the current assembly or from types that are derived from the containing class.

Is it possible to override a method which is declared without a virtual keyword?

Да, абстактный

What is the difference between new and override keywords in method declaration?

Переопределение: когда метод базового класса переопределяется в производном классе, используется версия в производном классе, даже если вызывающий код не «знал», что объект является экземпляром производного класса.

New: Если вы используете ключевое слово new вместо override, метод в производном классе не переопределяет метод в базовом классе, он просто скрывает его.

Если вы не укажете ни new, ни overrides, результирующий вывод будет таким же, как если бы вы указали new, но вы также получите предупреждение компилятора (поскольку вы можете не знать, что вы скрываете метод в базовом классе). или, возможно, вы захотели переопределить его и просто забыли включить ключевое слово).

Переопределение: используется с методом виртуального / абстрактного / переопределения в базовом классе.

Новое: когда базовый класс не объявил метод как виртуальный / абстрактный / переопределенный