**C# Tutorials**

**Difference between public, static and void**:

**public** :

The keyword public is an access modifier that tells the C# compiler that the Main method is accessible by anyone.

**static** :

The keyword static declares that the Main method is a global one and can be called without creating an instance of the class. The compiler stores the address of the method as the entry point and uses this information to begin execution before any objects are created.

**void** :

The keyword void is a type modifier that states that the Main method does not return any value.

**Object:**

An object is an instance of a class through which we access the methods of that class. "New" keyword is used to create an object. A class that creates an object in memory will contain the information about the methods, variables, and behavior of that class.

**Constructors:**

A constructor is a member function in a class that has the same name as its class. The constructor is automatically invoked whenever an object class is created. It constructs the values of data members while initializing the class.

**Difference between ref & out parameters:**

Both are used for the passing the arguments to methods as a reference type.

Both *ref*and *out*parameter treated same at compile-time but different at run-time.

An argument passed as ref must be initialized before passing to the method whereas out parameter needs not to be initialized before passing to a method.

**The use of 'using' statement in C#:**

The 'using' block is used to obtain a resource and process it and then automatically dispose of when the execution of the block completed.

**Difference between var & dynamic keyword:**

|  |  |
| --- | --- |
| **Var** | **Dynamic** |
| It is introduced in C# 3.0. | It is introduced in C# 4.0 |
| The variables are declared using var keyword are statically typed. | The variables are declared using dynamic keyword are dynamically typed. |
| The type of the variable is decided by the compiler at compile time. | The variable of this type need not be initialized at the time of declaration. Because the compiler does not know the type of the variable at compile time. |
| If the variable does not initialized it throw an error. | If the variable does not initialized it will not throw an error. |
| It support intelliSense in visual studio. | It does not support intelliSense in visual studio |
| var myvalue = 10; // statement 1 myvalue = “GeeksforGeeks”; // statement 2 Here the compiler will throw an error because the compiler has already decided the type of the myvalue variable using statement 1 that is an integer type. When you try to assign a string to myvalue variable, then the compiler will give an error because it violating safety rule type. | dynamic myvalue = 10; // statement 1 myvalue = “GeeksforGeeks”; // statement 2  Here, the compiler will not throw an error though the type of the myvalue is an integer. When you assign a string to myvalue it recreates the type of the myvalue and accepts string without any error. |
| It cannot be used for properties or returning values from the function. It can only used as a local variable in function. | It can be used for properties or returning values from the function. |

**Serialization:**

When we want to transport an object through a network, then we have to convert the object into a stream of bytes.

The process of converting an object into a stream of bytes is called Serialization. For an object to be serializable, it should implement ISerialize Interface.

De-serialization is the reverse process of creating an object from a stream of bytes.

**Can we use "this" command within a static method?**

We can't use 'This' in a static method because we can only use static variables/methods in a static method.

Why because static method does not need any object to be called, and ‘this’ keyword always point to a current object of a class.

**Difference between constant and readonly keyword:**

|  |  |
| --- | --- |
| **Readonly** | **Constant** |
| In C#, readonly fields can be created using readonly keyword | In C#, constant fields are created using const keyword. |
| ReadOnly is a runtime constant. | Const is a compile time constant. |
| The value of readonly field can be changed. | The value of the const field can not be changed. |
| It cannot be declared inside the method. | It can be declared inside the method. |
| In readonly fields, we can assign values in declaration and in the contructor part. | In const fields, we can only assign values in declaration part. |
| It can be used with static modifiers. | It cannot be used with static modifiers. |

**Difference between class and struct:**

|  |  |
| --- | --- |
| **Class** | **Struct** |
| Classes are of reference types. | Structs are of value types. |
| All the reference types are allocated on heap memory. | All the value types are allocated on stack memory. |
| Allocation of large reference type is cheaper than allocation of large value type. | Allocation and de-allocation is cheaper in value type as compare to reference type. |
| Class has limitless features. | Struct has limited features. |
| Class is generally used in large programs. | Struct are used in small programs. |
| Classes can contain constructor or destructor. | Structure does not contain constructor or destructor. |
| Classes used new keyword for creating instances. | Struct can create an instance, without new keyword. |
| A Class can inherit from another class. | A Struct is not allowed to inherit from another struct or class. |
| The data member of a class can be protected. | The data member of struct can’t be protected. |
| Function member of the class can be virtual or abstract. | Function member of the struct cannot be virtual or abstract. |
| Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable. | Each variable in struct contains its own copy of data(except in ref and out parameter variable) and any operation on one variable can not effect another variable. |

**Difference between Abstract class and Interface:**

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| It contains both declaration and definition part. | It contains only a declaration part. |
| Abstract class does not achieve multiple inheritance. | Multiple inheritance is achieved by interface. |
| It contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). | It does not contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). |
| It can contain static members. | It does not contain static members. |
| It can contain different types of access modifiers like public, private, protected etc. | It only contains public access modifier because everything in the interface is public. |
| The performance of an abstract class is fast. | The performance of interface is slow because it requires time to search actual method in the corresponding class. |
| It is used to implement the core identity of class. | It is used to implement peripheral abilities of class. |
| A class can only use one abstract class. | A class can use multiple interface. |
| If many implementations are of the same kind and use common behavior, then it is superior to use abstract class. | If many implementations only share methods, then it is superior to use Interface. |
| Abstract class can contain methods, fields, constants, etc. | Interface can only contain methods . |
| It can be fully, partially or not implemented. | It should be fully implemented. |

**Value Type:**

A Value Type holds the data within its own memory allocation.

Value Type variables are stored in the stack memory.

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 Type contains a pointer to another memory location that holds the real data.

Reference Type variables are stored in the heap memory.

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.

**Sealed Class**

Sealed classes are used to restrict the users from inheriting the class. A class can be sealed by using the *sealed* keyword.

The keyword tells the compiler that the class is sealed, and therefore, cannot be extended. No class can be derived from a sealed class.

*A method can also be sealed*, and in that case, the method cannot be overridden. However, a method can be sealed in the classes in which they have been inherited. If you want to declare a method as sealed, then it has to be declared as **virtual** in its base class.

**Method overloading:**

Method overloading is creating multiple methods with the same name with unique signatures in the same class. When we compile, the compiler uses overload resolution to determine the specific method to be invoke.

**Difference between Array and Arraylist:**

**Arrays:** An array is a group of like-typed variables that are referred to by a common name.

**ArrayList:**ArrayList represents an ordered collection of an object that can be indexed individually. It is basically an alternative to an array. It also allows dynamic memory allocation, adding, searching and sorting items in the list.

|  |  |  |
| --- | --- | --- |
| **FEATURE** | **ARRAY** | **ARRAYLIST** |
| Memory | This has fixed size and can’t increase or decrease dynamically. | Size can be increase or decrease dynamically. |
| Namespace | Arrays belong to **System.Array**namespace | ArrayList belongs to **System.Collection**namespace. |
| Data Type | In Arrays, we can store only one datatype either int, string, char etc. | In ArrayList, we can store different datatype variables. |
| Operation Speed | Insertion and deletion operation is fast. | Insertion and deletion operation in ArrayList is slower than an Array. |
| Typed | Arrays are strongly typed which means it can store only specific type of items or elements. | Arraylist are not strongly typed. |
| null | Array cannot accept null. | ArrayList can accepts null. |
| Memory | This has fixed size and can’t increase or decrease dynamically. | Size can be increase or decrease dynamically. |

**Access Modifiers**:

Access modifiers specify the accessibility of an object and all of its members in the C# project

**Type**:

* Private
* Protected
* Public
* Internal
* Protected Internal

**Private Access Modifier**

Objects that implement **private** access modifier are accessible only inside a class or a structure. As a result, we can’t access them outside the class they are created

**Public Access Modifier**

Objects that implement **public** access modifier are accessible from everywhere in our project. Therefore, there are no accessibility restrictions

**Protected Access Modifier**

The **protected** keyword implies that the object is accessible inside the class and in all classes that derive from that class.

**Internal Access Modifier**

The internal keyword specifies that the object is accessible only inside its own assembly but not in other assemblies

**Protected Internal Access Modifier**

The **protected** **internal**access modifier is a combination of protected and internal. As a result, we can access the protected internal member only in the same assembly or in a derived class in other assemblies (projects)

Default Access modifiers:

|  |  |  |
| --- | --- | --- |
| **Member** | **Default Access modifier** | **Comments** |
| Namespaces | public | No access modifiers |
| Class | internal | can have public or internal |
| Class members | private | can have any of the five kinds of declared accessibility |
| Struct members | private | can have public, internal, or private declared accessibility |
| Interface | public | No access modifiers are allowed on interface member declarations |
| Enumeration | public | No access modifiers are allowed on enumeration member declarations. |

What's the difference between the System.Array.CopyTo() and System.Array.Clone() ?

Using Clone() method, we creates a new array object containing all the elements in the original Array and using CopyTo() method. All the elements of existing array copies into another existing array. Both methods perform a shallow copy.

**Finally keyword**:

The finally block will execute when the try/catch block leaves the execution, no matter what condition cause it. **It always executes whether the try block terminates normally or terminates due to an exception.**

The main purpose of finally block is to release the system resources. The finally block follows try/catch block.

Notes:

* In C#, multiple finally blocks in the same program are not allowed.
* The finally block does not contain any return, continue, break statements because it does not allow controls to leave the finally block.
* You can also use finally block only with a try block means without a catch block but in this situation, no exceptions are handled.
* The finally block will be executed after the try and catch blocks, but before control transfers back to its origin.