C# Static Keyword

In c#, **static** is a keyword or a modifier which is useful to make a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) or [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) or variable properties as not instantiable that means we cannot instantiate the items which we declared with a static modifier.

The **static** members which we declared can be accessed directly with a type name. Suppose if we apply a static modifier to a class property or to a method or variable, then we can access those static members directly with a class name, instead of creating an object of a class to access those properties.

## C# Static Variables

Following is the example of defining a class with static properties and those can be accessed directly with the type instead of a specific object name.

class User

{

public static string name, location;

public static int age;

}

If you observe the above example, we defined variables with static keyword and we can access those variables directly with a type name like **User.name** or **User.location** and **User.age**.

Following is the example of accessing the variables directly with a type name in c# programming language.

Console.WriteLine(User.name);

Console.WriteLine(User.location);

Console.WriteLine(User.age);

If you observe above statements, we are accessing our **static** properties directly by using class name instead of with class instance.

Generally, in c# the instance of a class will contain a separate copy of all instance fields so the memory consumption will increase automatically, but if we use static modifier there is only one copy of each field so automatically the memory will be managed efficiently.

In c#, we can use static modifier with [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples), [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples), properties, [constructors](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples), [operators](https://www.tutlane.com/tutorial/csharp/csharp-operators-arithmetic-relational-logical-assignment-precedence), fields and with events but it cannot be used with **indexers**, **finalizers** or types other than [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples).

## C# Static Keyword Example

Following is the example of creating a class by including both static and non-static variables & methods. Here we can access non-static variables and methods by creating an instance of the class, but it’s not possible for us to access the static fields with an instance of the class so the static variables and methods can be accessed directly with the class name.

using System;

namespace YourApp

{

class User

{

// Static Variables

public static string name, location;

//Non Static Variable

public int age;

// Non Static Method

public void Details()

{

Console.WriteLine("Non Static Method");

}

// Static Method

public static void Details1()

{

Console.WriteLine("Static Method");

}

}

class Program

{

static void Main(string[] args)

{

User u = new User();

u.age = 32;

u.Details();

User.name = "Suresh Dasari";

User.location = "Hyderabad";

Console.WriteLine("Name: {0}, Location: {1}, Age: {2}", User.name, User.location, u.age);

User.Details1();

Console.WriteLine("\nPress Enter Key to Exit..");

Console.ReadLine();

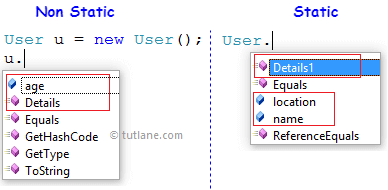
}

}

}

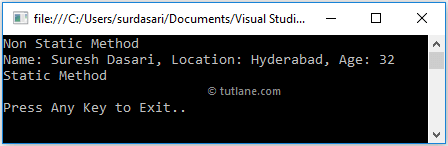
If you observe above example, we created a class called “**User**” with **static** and **non-static** variables & methods. Here we are accessing **non-static** variables and methods with an instance of **User** class and **static** fields & methods are able to access directly with the class name (**User**).

The following diagram will illustrate more details about how **static** and **non-static** variables & methods can be accessed in our c# application.



If you observe the above diagram, it clearly says that **non-static** fields and methods can be accessed only with an instance of the class and the **static** fields & methods can be accessed directly with the class name.

When you run the above c# program, you will get the result as shown below.



This is how you can use static keyword in our c# applications to make a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) or [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) or variable properties as not instantiable based on our requirements.

# C# Static Class with Examples

In c#, a **static class** can be created by using static modifier and the static class can contain only static members.

Generally, the **static class** is same as **non-static** **class**, but the only difference is the **static class** cannot be instantiated. Suppose if we apply static modifier to a class, then we should not use the **new** keyword to create a variable of the class type.

Another difference is the **static class** will contain only [static](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword) members, but the **non-static class** can contain both [static](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword) and non-static members.

## C# Static Class Syntax

In c#, we can create a **static class** by applying static keyword to the class like as shown below.

static class sample

 {

     //static data members

    //static methods

 }

If you observe the above syntax, to create a static class called “**sample**”, we applied static keyword to the class type. The methods and data members that we are going to implement in the sample class must be **static**.

In c#, we can access members of a static class directly with the class name. For example, we have a static class called “**User**” with a method “**Details()**” that we can access like **User.Details()**.

## C# Static Class Example

Following is the example of defining a **static class** to access data members and member functions without creating an instance of the class in c# programming language.

using System;

namespace YourApp

{

    static class User

    {

        // Static Variables

        public static string name;

        public static string location;

        public static int age;

        // Static Method

        public static void Details()

        {

            Console.WriteLine("Static Method");

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            User.name = "Suresh Dasari";

            User.location = "Hyderabad";

            User.age = 32;

            Console.WriteLine("Name: {0}", User.name);

            Console.WriteLine("Location: {0}", User.location);

            Console.WriteLine("Age: {0}", User.age);

            User.Details();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

        }

    }

}

If you observe above example, we are accessing **static** class members and functions directly with the class name because we cannot instantiate the **static class**.

Try executing the program.

This is how we can create a static class and use it in our c# applications based on our requirements.

## C# Static Class Features

Following are the main features of static class in c# programming language.

* The static class in c# will contain only **static** members.
* In c#, the static classes cannot be instantiated.
* C# static classes are **sealed**, therefore it cannot be inherited.
* The static classes in c# will not contain [instance constructors](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples).

As discussed in the previous article [static keyword in c#](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword), we can use **static** members in **non-static**[classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) such as normal [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples). For normal [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples), you can create an instance of class using the **new** keyword to access non-static members and functions but it cannot access the **static** members and [functions](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples).

To know more about it, check this article [static keyword in c# with examples](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword).

The advantage of using static classes in c# applications will make sure that instance of classes cannot be created.

# C# Readonly Property

In c#, **readonly** is a keyword which is useful to define read-only fields in our applications.  The read-only field values need to be initialized either at the declaration or in a constructor of the same class unlike [constant keyword in c#](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword). If we use readonly keyword with fields, then those field values will be evaluated at the runtime.

To define read-only fields in c#, we need to use readonly keyword during the declaration of fields in our application and we can use readonly modifier with the numbers, boolean values, strings or with null references.

In c#, if we use readonly keyword to define the read-only field, then that field value cannot be changed once the [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples) execution has finished so we should not use readonly keyword with the fields whose value will be changed at any time.

It’s mandatory to initialize read-only field values either at the declaration or in a [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples) otherwise we will get compile-time errors in our c# application.

## C# Readonly Keyword Syntax

Following is the syntax of defining read-only fields using readonly keyword in c# programming language.

readonly data\_type field\_name = "value";

If you observe the above syntax, we used a readonly keyword to declare a read-only variable in our application.

The following are the different ways of declaring and initializing read-only fields in c# programming language.

class User {

        // Initialize Read Only Fields

        public readonly string name="Suresh Dasari";

        public readonly string location;

        public readonly int age;

        public User() {

            location = "Hyderabad";

            age = 32;

        }

        public void SetDetails() {

            // Compile error if uncommented

            //location = "Guntur";

            //age = 30;

        }

    }

If you observe above example, we created a read only fields with different [data types](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) and initializing field values during the declaration and in a [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples).

In case if we uncomment the commented code in **SetDetails()** method, then we will get compile errors because, in c#, the read-only field values can be initialized either during declaration or in a [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples).

## C# Readonly Property Example

Following is the example of defining and using read-only fields in c# programming language with readonly keyword.

using System;

namespace YourApp

{

    class User

    {

        // Initialize Read Only Fields

        public readonly string name = "Suresh Dasari";

        public readonly string location;

        public readonly int age;

        public User()

        {

            location = "Hyderabad";

            age = 32;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            User u = new User();

            // This will throw compile time error

            //u.name = "Rohini Alavala";

            Console.WriteLine("Name: {0}", u.name);

            Console.WriteLine("Location: {0}", u.location);

            Console.WriteLine("Age: {0}", u.age);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

        }

    }

}

If you observe above example, we created a read only fields with different [data types](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) and if we uncomment the commented code, then we will get a compile time error because we are trying to change the value of read only fields.

As discussed, once the read-only field is declared and initialized, then that field value must be same throughout the application.

Try executing the above program.

## C# ReadOnly Property Features

The following are the important features of the read-only variables in c# programming language.

* Read-only fields in c# can be created by using readonly keyword.
* In c#, the readonly fields can be initialized either at the declaration or in a constructor.
* The readonly field values will be evaluated during the run time in c#.
* Once values assigned to read-only fields, then those values must be the same throughout the application.

## C# Difference between Constant and ReadOnly

The following are the difference between [constant](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword) and readonly properties in c# programming language.

* In c#, the [constant](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword) fields can be created by using const keyword and read-only fields can be created by using readonly keyword.
* In c#, the [constant](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword) fields can only be initialized during the time of declaration but read-only fields can be initialized either at the declaration or in a [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples).
* Constant field values will be evaluated during the compile time but read-only field values will be evaluated at run time in c#.

This is how we can create and use read-only fields in our c# programming language with readonly keyword based on our requirements.

# C# const (Constant) Keyword

In c#, **const** is a keyword that is useful to declare constant fields in our applications. Generally, in c# the constant field values are set at compile-time and those values will never change.

In c#, if we use const keyword to declare a constant field, then that field value cannot be changed throughout the application so we should not use const keyword with the fields whose value will be changed at any time.

To define constant fields in c#, we need to use const keyword during the declaration of fields in our application and we can use constants with numbers, boolean values, strings or with null references.

It’s mandatory to initialize constant fields with required values during the declaration itself otherwise we will get compile-time errors in our c# application.

In c#, the [static modifier](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword) is not allowed to use during the declaration of constant variables.

## C# Constant Syntax

Following is the syntax of defining constant fields using const keyword in c# programming language.

const data\_type field\_name = "value";

If you observe the above syntax, we used a const keyword declare a constant variable in our application.

The following are the different ways of declaring and initializing a constant variables in c# programming language.

// Constant variables

const string name = "Suresh Dasari";

const string location = "Hyderabad";

const int age = 32;

If you observe above examples, we created a constant fields with different [data types](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) based on our requirements.

## C# Constant Example

Following is the example of defining and using constant fields in c# programming language with const keyword.

 using System;

namespace YourApp

{

    class Program

    {

        static void Main(string[] args)

        {

            // Constant variables

            const string name = "Suresh Dasari";

            const string location = "Hyderabad";

            const int age = 32;

            // This will throw compile time error

            //name = "Rohini Alavala";

            Console.WriteLine("Name: {0}", name);

            Console.WriteLine("Location: {0}", location);

            Console.WriteLine("Age: {0}", age);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

        }

    }

}

If you observe above example, we created a constant fields with different [data types](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) and if we uncomment the commented code, then we will get a compile time error because we are trying to change the value of constant fields.

As discussed, once the constant field is declare and initialized, then that field value must be same throughout the application.

Try executing the above program.

## C# Constant Features

The following are the important features of a constant variable in c# programming language.

* Constant fields in c# can be created by using const keyword.
* In c#, the constant fields must be initialized during the time of declaration.
* Constant field values will be evaluated during the compile time in c#.
* Once values assigned to constant fields, then those values must be the same throughout the application.

This is how we can use constant fields in our c# programming language based on our requirements.

# C# Enum (Enumerator)

In c#, **enum** is a keyword which is used to declare an enumeration. In c#, the enumeration is a type that consists of a set of named [constants](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword) as a list.

By using an enumeration, we can group constants that are logically related to each other. For example, the days of the week can be grouped together by using enumeration in c#.

## C# Enum Syntax

Following is the syntax of defining an enumeration using enum keyword in c# programming language.

enum enum\_name

{

// enumeration list

}

If you observe the above syntax, we used enum keyword to define an enumeration based on our requirements.

Following is the example of defining an enumeration using enum keyword in c# programming language.

enum Week

{

    Sunday,

    Monday,

    Tuesday,

    Wednesday,

    Thursday,

    Friday,

    Saturday

}

If you observe the above example, we define an enumeration “**Week**” with a list of named [constants](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword) called enumeration list.

In c# by default, the first-named constant in enumerator has a value of **0**, and the value of each successive item in enumerator will be increased by **1**. For example, in the above enumeration, Sunday value is **0**, Monday is **1**, Tuesday is **2**, and so forth.

In case, if we want to change the default values of an enumerator, then assigning a new value to the first item in enumerator will automatically assign incremental values to the successive items in an enumerator.

Following is the example of overriding the default values of an enumerator by assigning a new value to the first item in an enumerator.

enum Week

  {

      Sunday = 10,

      Monday,

      Tuesday,

      Wednesday,

      Thursday,

      Friday,

      Saturday

  }

In the above enumeration, the sequence of elements is forced to start from **10** instead of **0** like Sunday value is **10**, Monday is **11**, Tuesday is **12**, and so forth.

## C# Enum with Integer Types

In c#, an enumeration can contain only integral data type items such as **byte**, **sbyte**, **short**, **ushort**, **int**, **uint**, **long** or **ulong**. It’s not possible to use an enum with **string** or any other data types to define enum elements except numeric types.

The default type of enumeration element is **int** (integer). In case, if you want to change the integral type of enum to **byte**, then you need to mention a **byte** type with a colon (**:**) after the identifier like as shown below.

enum Week : byte

  {

      Sunday,

      Monday,

      Tuesday,

      Wednesday,

      Thursday,

      Friday,

      Saturday

  }

If you observe the above example, we are trying to change the default integral type of elements in an enumeration to **byte** type.

To get the values of enum elements in c#, an explicit cast is necessary to convert from **enum** type to an integral type.

For example, the following are the statements to get an enum item value by using a cast to convert from **enum** to **int**.

int a = (int)Week.Sunday; // It will return 0

int b = (int)Week.Monday; // It will return 1

If you observe above statements, we are doing cast conversion (**enum** to **int**) to get an enum item values.

## C# Enum Example

Following is the example of declaring an enumeration using enum keyword in c# programming language.

using System;

namespace YourApp

{

    class Program

    {

        enum Week

        {

            Sunday,

            Monday,

            Tuesday,

            Wednesday,

            Thursday,

            Friday,

            Saturday

        }

        static void Main(string[] args)

        {

            int a = (int)Week.Sunday;

            int b = (int)Week.Monday;

            int c = (int)Week.Tuesday;

            Console.WriteLine(Week.Sunday);

            Console.WriteLine(Week.Monday);

            Console.WriteLine("Sunday: {0}", a);

            Console.WriteLine("Monday: {0}", b);

            Console.WriteLine("Tuesday: {0}", c);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

        }

    }

}

If you observe above example, we defined an enumeration **Week** and getting the values of enumeration items by explicitly converting from **enum** to **int** and assigned to an integer variables.

Try executing the above program.

If you observe the above result, the first enumerator (**Sunday**) has a value of **0** and the value of each successive enumerator is increased by **1**.

## C# Enum Methods

In c#, we have a class called **Enum** that contains the following helper methods to work with an enumeration (enum).

| **Method** | **Description** |
| --- | --- |
| Format | It is useful to convert the value of enum type to a specified string format. |
| GetName | It is useful to get the name of the specified enum item. |
| GetNames | It is useful to get all item names of the specified enum as an array. |
| GetValues | It is useful to get all item values of the specified enum as an array. |
| Parse | It is useful to convert the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object. |
| GetUnderlyingType | It is useful to return the underlying type of the specified enumeration. |

## C# Iterate through Enum

In c#, we can iterate or loop through enum items using [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) or [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) loop to get the enumeration item names or values by using enum helper methods.

Following is the example of a loop through enum items to get all item names and values using enum helper methods based on our requirements.

using System;

namespace YourApp

{

    class Program

    {

        enum Week

        {

            Sunday,

            Monday,

            Tuesday,

            Wednesday,

            Thursday,

            Friday,

            Saturday

        }

        static void Main(string[] args)

        {

            Console.WriteLine("Week Enumeration Values");

            foreach (string w in Enum.GetNames(typeof(Week)))

            {

                Console.WriteLine(w);

            }

            Console.WriteLine("\nPress Any Key to Exit..");

            Console.ReadLine();

        }

    }

}

If you observe above example, we are looping through an enumeration using [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \t "_blank" \o "C# Foreach Loop with Examples) loop and getting all item names using **GetNames()** helper method.

Try executing the above program.

If you observe the above result, we are able to get all enumeration item names by looping through an enumeration using a [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) loop based on our requirements.

In c#, it’s better to define an enum within a [namespace](https://www.tutlane.com/tutorial/csharp/csharp-namespaces-with-examples) so that all the [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) in the namespace can access equally and we can also nested an enum within [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) or [struct](https://www.tutlane.com/tutorial/csharp/csharp-structures-structs" \o "C# Strutures with Examples" \t "_blank) based on our requirements.