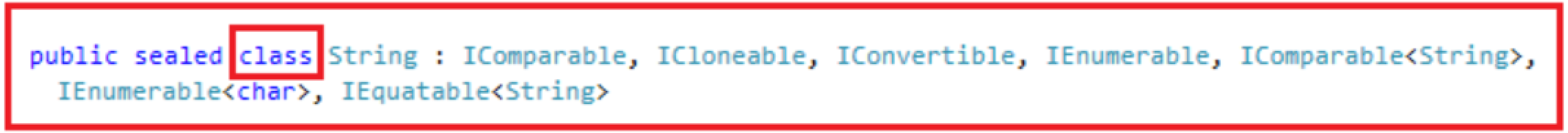
C# String in Depth with Examples

Strings are reference types in C#:

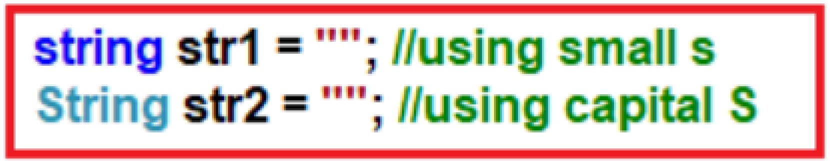
Then if you right-click on the string data type and click on go to definition then you will see that it is a class. Class means reference data type.



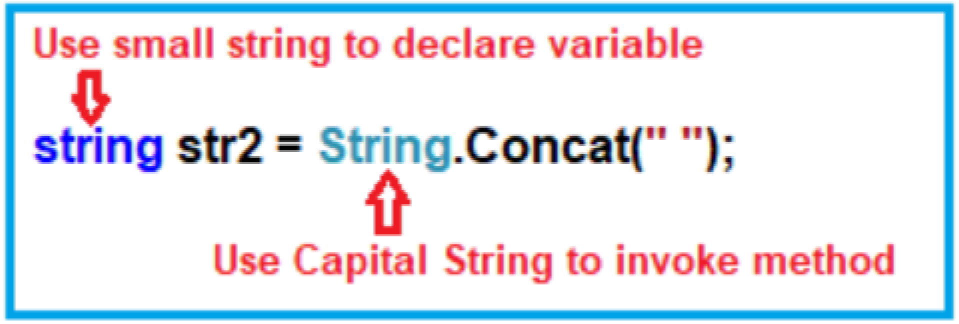
```
public sealed class String : IComparable, ICloneable, IConvertible, IEnumerable, IComparable<String>,
  IEnumerable<char>, IEquatable<String>
```

What are the Differences between String(Capital) vs string(small) in C#?

In C#, you can use the string in two ways i.e. you can use the string using capital S (i.e. String) or by using the small "s" (i.e. string) as shown in the below image. The small string is actually an alias of String (Capital string).



```
string str1 = ""; //using small s
String str2 = ""; //using capital S
```

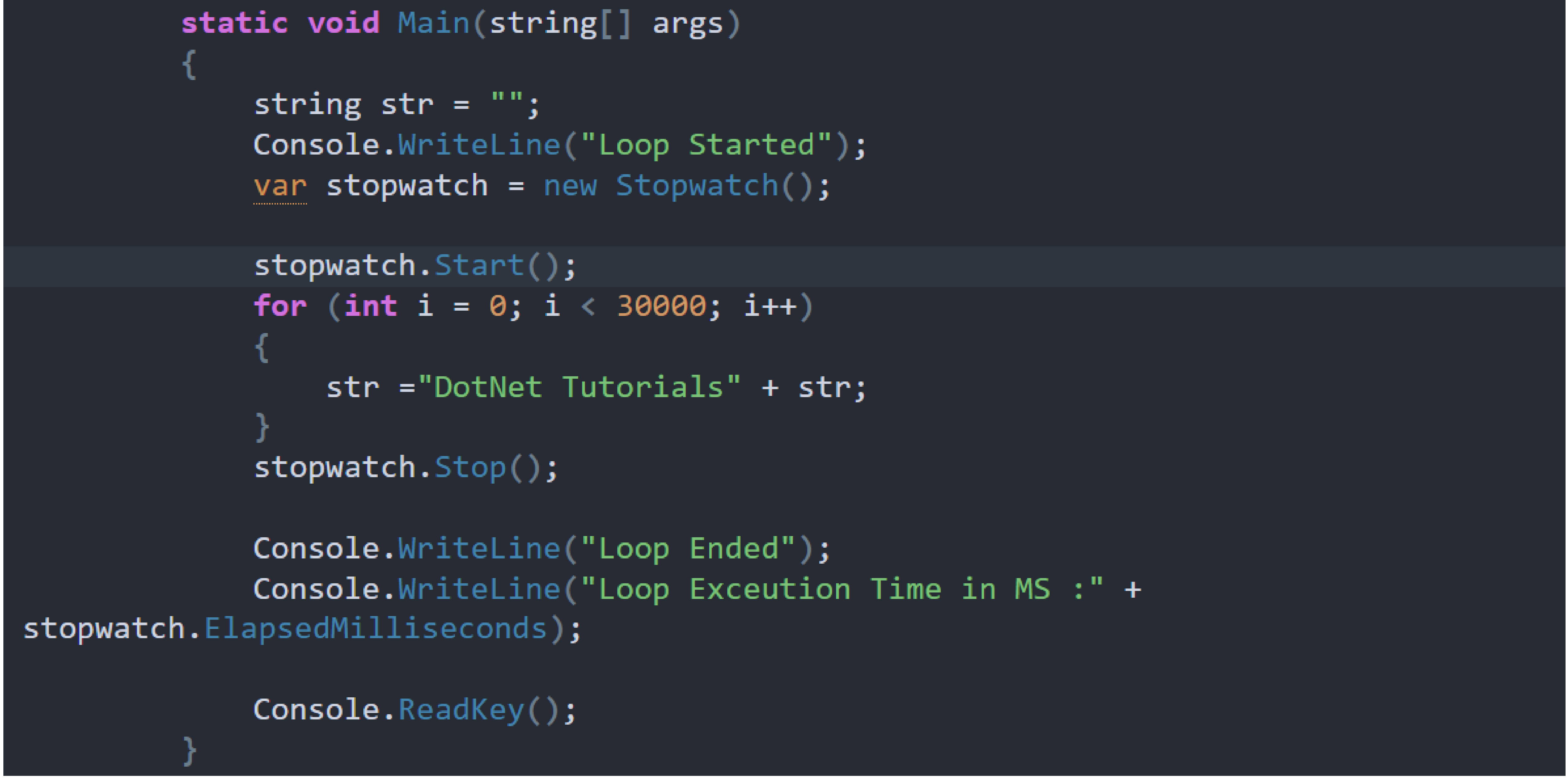


Use small string to declare variable
Use Capital String to invoke method

```
string str2 = String.Concat(" ");
```

Strings are Immutable in C#:

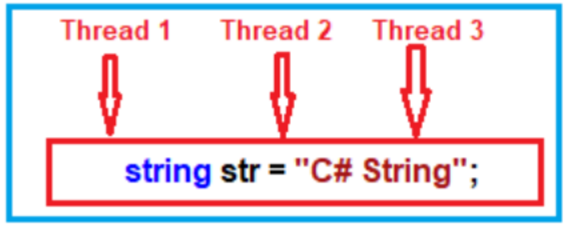
Before understanding strings are immutable, first, we need to understand two terms i.e. Mutable and Immutable. Mutable means can be changed whereas Immutable means can not be changed. C# strings are immutable means C# strings cannot be changed



```
        static void Main(string[] args)
        {
            string str = "";
            Console.WriteLine("Loop Started");
            var stopwatch = new Stopwatch();

            stopwatch.Start();
            for (int i = 0; i < 30000; i++)
            {
                str ="DotNet Tutorials" + str;
            }
            stopwatch.Stop();

            Console.WriteLine("Loop Ended");
            Console.WriteLine("Loop Exceution Time in MS :" +
 stopwatch.ElapsedMilliseconds);

            Console.ReadKey();
        }
```

Why they made C# String Immutable?

They made Strings as Immutable for Thread Safety. Think of one situation where you have many threads and all the threads want to manipulate the same string object as shown in the below image. If strings are mutable then we have thread-safety issues.



```
string str = "C# String";
```

What is an Array in C#?

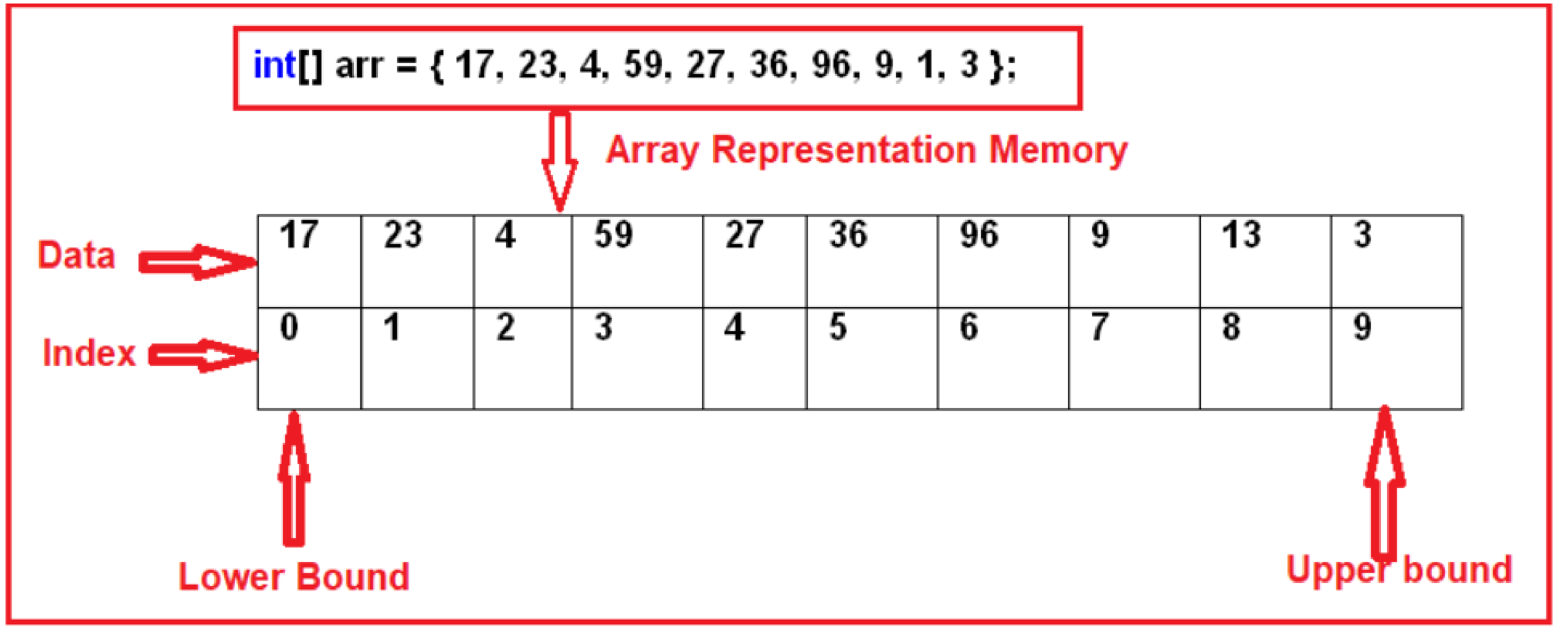
In simple words, we can define an array as a collection of similar types of values that are stored in sequential order i.e. they are stored in a contiguous memory location.

Single dimensional array

Multi-dimensional array

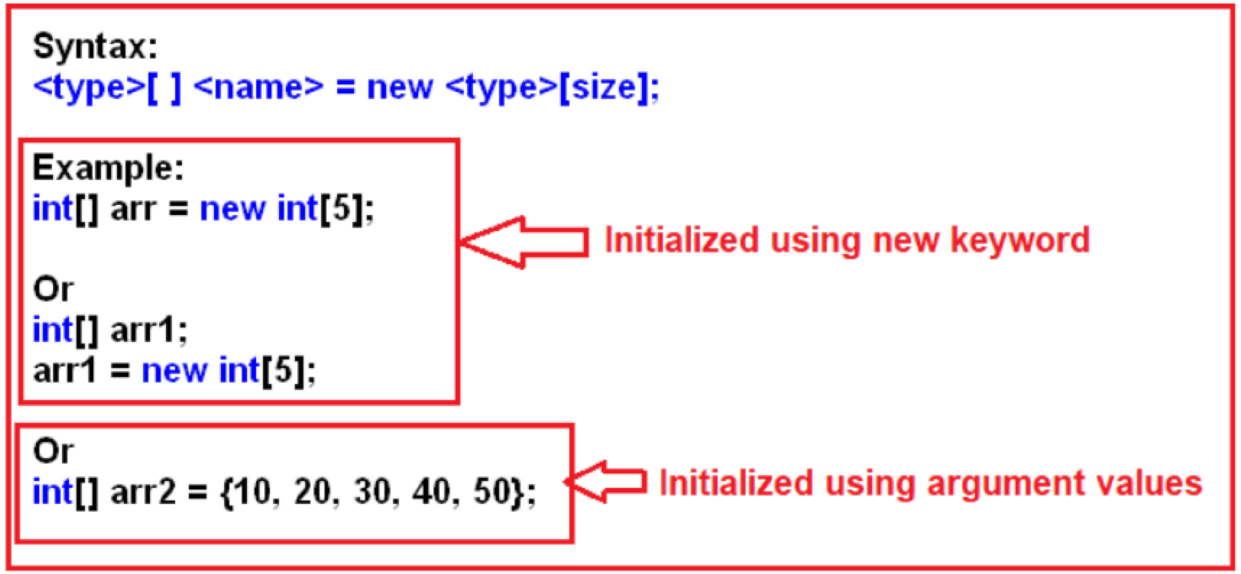
1.Jagged array: Whose rows and columns are not equal
2.Rectangular array: Whose rows and columns are equal

Memory Representation of Arrays in C#:


```
int[] arr = { 17, 23, 4, 59, 27, 36, 96, 9, 1, 3 };
```

One Dimensional Array in C# with Examples:

The array which stores the data in the form of rows in a sequential order is called a one-dimensional array in C#. The syntax for creating a one-dimensional array in C# is given below.



```
Syntax:
<type>[ ] <name> = new <type>[size];

Example:
int[] arr = new int[5];

Or
int[] arr1;
arr1 = new int[5];

Or
int[] arr2 = {10, 20, 30, 40, 50};
```

Initialized using new keyword
Initialized using argument values

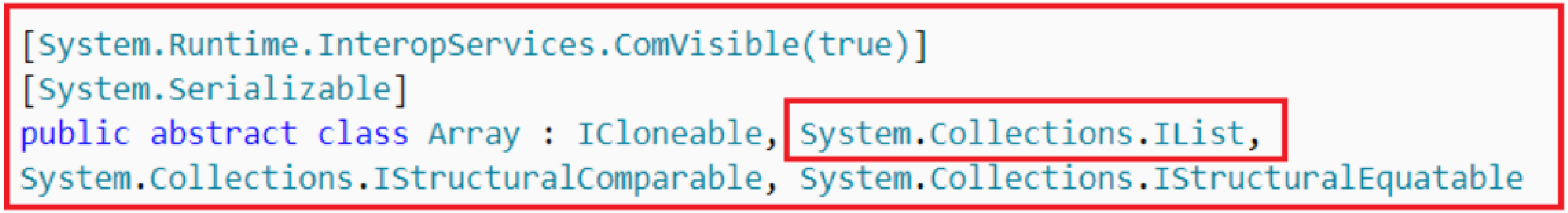
```
static void Main(string[] args)
{
//Creating an array with size 6
int[] arr = new int[6];
//accessing array values using loop
//Here it will display the default values
//as we are not assigning any values
for (int i = 0; i < 6; i++)
{
Console.Write(arr[i] + " ");
}
Console.WriteLine();
int a = 0;
//Here we are assigning values to array using for loop
for (int i = 0; i < 6; i++)
{
a += 10;
arr[i] = a;
}
//accessing array values using foreach loop
foreach (int i in arr)
{
Console.Write(i + " ");
}
Console.ReadKey();
}
```

For each loop in C#:

This for each loop is specially designed in C# for accessing the values from a collection like an array. When we use a for-each loop for accessing the values of an array or collection, we only require to hand over the array or collection to the loop which does not require any initialization, condition, or iteration. The loop itself starts its execution by providing access to each and every element present in the array or collection starting from the first up to the last element in sequential order.

What is the Array class in C#?

The **Array** class is a predefined class that is defined inside the **System** namespaces. This class is working as the base class for all the arrays in C#. The **Array** class provides a set of members (methods and properties) to work with the arrays such as creating, manipulating, searching, reversing, and sorting the elements of an array, etc. The definition of the Array class in C# is gen below.



```
[System.Runtime.InteropServices.ComVisible(true)]
[System.Serializable]
public abstract class Array : ICloneable, System.Collections.IList,
System.Collections.IStructuralComparable, System.Collections.IStructuralEquatable
```

- ✓ **Sort(<array>):** Sorting the array elements
- ✓ **Reverse (<array>):** Reversing the array elements
- ✓ **Copy (src, dest, n):** Copying some of the elements or all elements from the old array to the new array
- ✓ **GetLength(int):** A 32-bit integer that represents the number of elements in the specified dimension.
- ✓ **Length:** It Returns the total number of elements in all the dimensions of the Array; zero if there are no elements in the array.

Advantages of using an Array in C#:
The advantages of using an array in C# are as follows:
1.It is used to represent similar types of multiple data items using a single name.
2.We can use arrays to implement other data structures such as linked lists, trees, graphs, stacks, queues, etc.
3.The two-dimensional arrays in C# are used to represent matrices.
4.The Arrays in C# are strongly typed. That means they are used to store similar types of multiple data items using a single name. As the arrays are strongly typed so we are getting two advantages. First, the performance of the application will be much better because boxing and unboxing will not happen. Secondly, runtime errors will be prevented because of a type mismatch. In this case, at compile time it will give you the error if there is a type mismatch.

Disadvantages of using Arrays in C#:
1.The array size is fixed. So, we should know in advance how many elements are going to be stored in the array. Once the array is created, then we can never increase the size of an array. If you want then we can do it manually by creating a new array and copying the old array elements into the new array.
2.As the array size is fixed, if we allocate more memory than the requirement then the extra memory will be wasted. On the other hand, if we allocate less memory than the requirement, then it will create the problem.
3.We can never insert an element into the middle of an array. It is also not possible to delete or remove elements from the middle of an array.

What is a Two-Dimensional Array in C#?

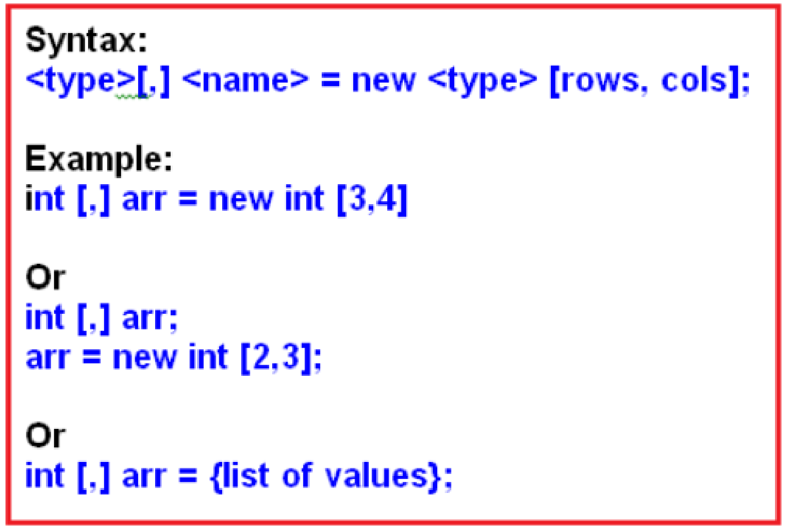
The arrays which store the elements in the form of rows and columns are called Two-Dimensional Array in C#. The two-dimensional array which is also called multidimensional array is of two types in C#. They are as follows

1.Rectangular array: The array whose rows and columns are equal are called a rectangular array

2.Jagged array: The array whose rows and columns are not equal are called a jagged array

Rectangular 2d Array in C#:

Let us first understand the syntax of the Two-Dimensional Array in C#. Please have a look at the following diagram.



Syntax:

```
<type>[,] <name> = new <type> [rows, cols];
```

Example:

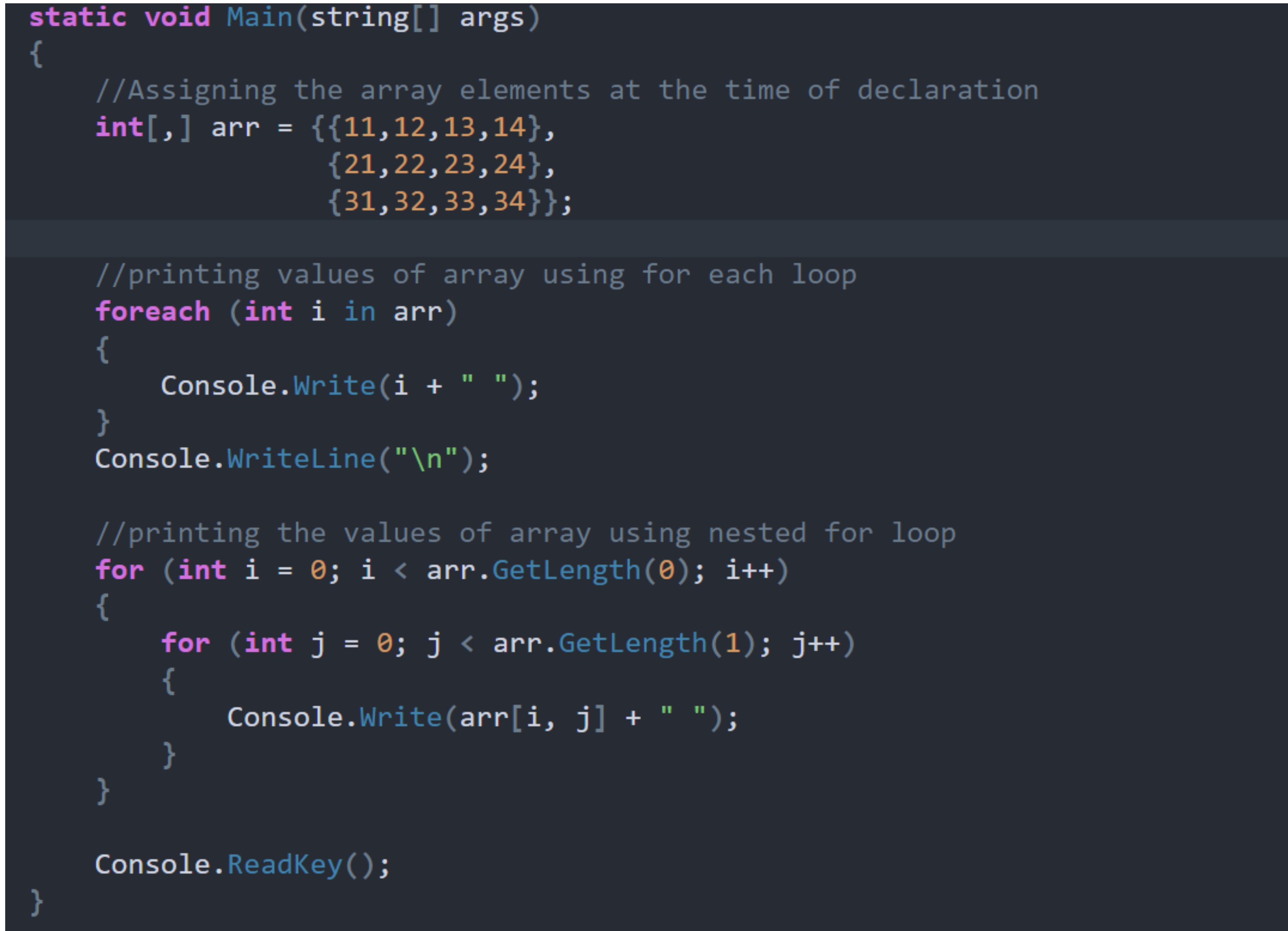
```
int [,] arr = new int [3,4]
```

Or

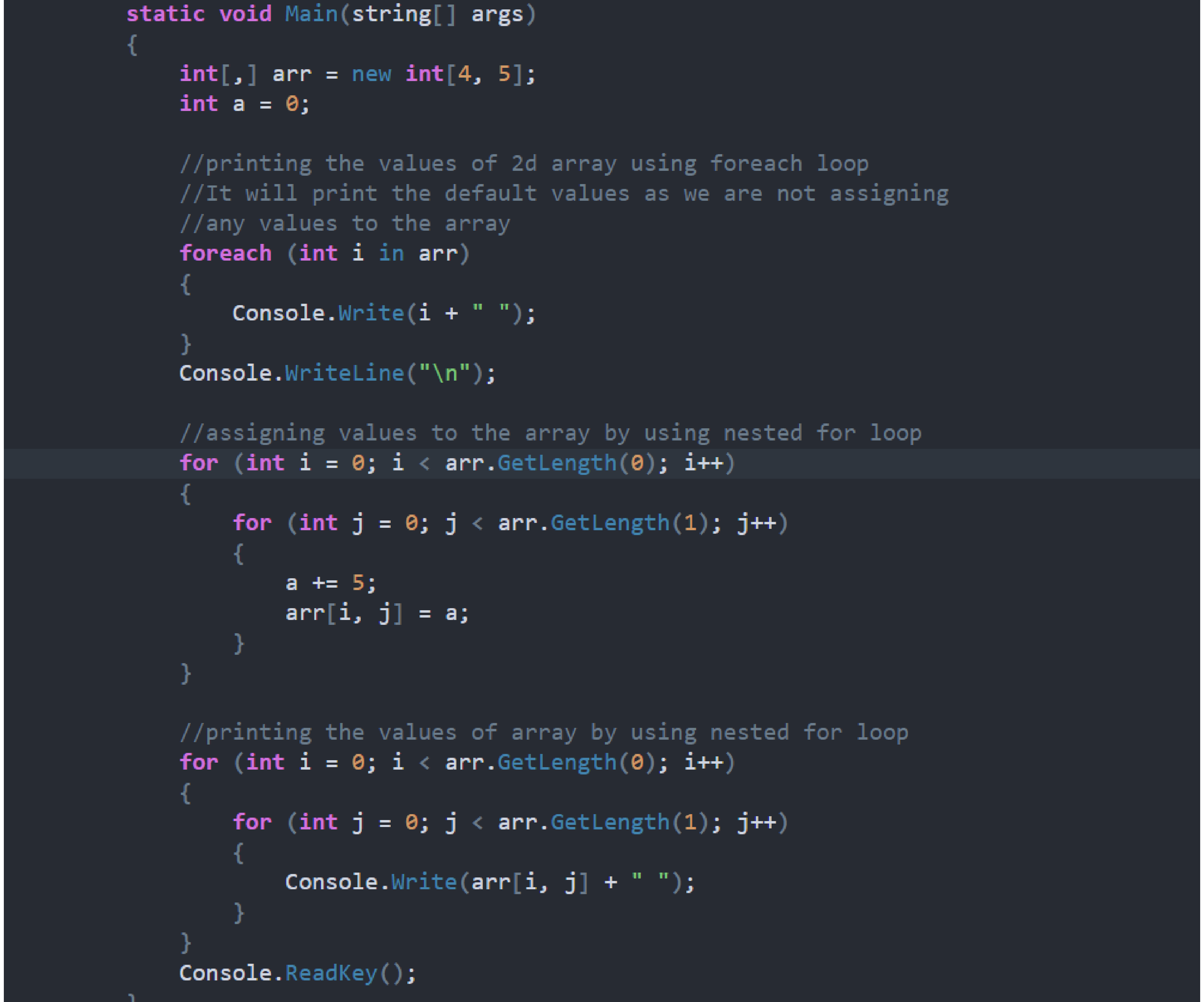
```
int [,] arr;
arr = new int [2,3];
```

Or

```
int [,] arr = {list of values};
```



```
static void Main(string[] args)
{
    //Assigning the array elements at the time of declaration
    int[,] arr = {{11,12,13,14},
                  {21,22,23,24},
                  {31,32,33,34}};

    //printing values of array using for each loop
    foreach (int i in arr)
    {
        Console.Write(i + " ");
    }
    Console.WriteLine("\n");

    //printing the values of array using nested for loop
    for (int i = 0; i < arr.GetLength(0); i++)
    {
        for (int j = 0; j < arr.GetLength(1); j++)
        {
            Console.Write(arr[i, j] + " ");
        }
    }

    Console.ReadKey();
}
```



```
        static void Main(string[] args)
        {
            int[,] arr = new int[4, 5];
            int a = 0;

            //printing the values of 2d array using foreach loop
            //It will print the default values as we are not assigning
            //any values to the array
            foreach (int i in arr)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine("\n");

            //assigning values to the array by using nested for loop
            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    a += 5;
                    arr[i, j] = a;
                }
            }

            //printing the values of array by using nested for loop
            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    Console.Write(arr[i, j] + " ");
                }
            }
            Console.ReadKey();
        }
```