Lecture No 3

Muhammad Siddique

Chapters Included: 7,10

Outline

- Define a class containing a related set of methods and data items.
- Control the accessibility of members by using the public and private keywords.
- Create objects by using the new keyword to invoke a constructor.
- Create class using static keyword.
- Anonymous classes.
- Arrays.
- Strings Handling.

Defining and using a class

- In C#, you use the class keyword to define a new class.
- The data and methods of the class occur in the body of the class between a pair of braces.
- Following is a C# class called Circle that contains one method (to calculate the circle's area) and one piece of data (the circle's radius): e.g.

```
class Circle  {
int radius;
double Area()
{ return Math.PI * radius * radius;  }
}
```

Controlling Accessibility

- Fields (such as radius) and methods (such as Area) defined in the class can be used by other methods inside the class but not by the outside world because they are private to the class.
- You have to modify the definition of a field or method with the public and private keywords to control whether it is accessible from the outside:
  - A method or field is private if it is accessible only from within the class. To declare that a method or field is private, you write the keyword private before its declaration.
  - A method or field is public if it is accessible both within and from outside the class. To declare that a method or field is public, you write the keyword public before its declaration.

Controlling Accessibility

- Now see the Circle class again.
- This time, Area is declared as a public method and radius is declared as a private field:

```
class Circle  {
Private int radius;
Public double Area()
{
return Math.PI * radius * radius;
}
}
```

Working with Constructors

- When you use the new keyword to create an object, the runtime needs to construct that object by using the definition of the class.
- The runtime must grab a piece of memory from the operating system, fill it with the fields (variable) defined by the class, and then invoke a constructor to perform any initialization required.
- A constructor is a special method that runs automatically when you create an instance (Object) of a class.
- It has the same name as the class, and it can take parameters, but it cannot return a value (not even void).
- Every class must have a constructor.

Working with Constructors

- If you don’t write one, the compiler automatically generates a default constructor for you.
- Just add a public method that does not return a value and give it the same name as the class.
- The following example shows the Circle class with a default constructor that initializes the radius field to 0:

```
class Circle
{
private int radius;
public Circle() // default constructor
{
radius = 0;
}

public double Area()
{
return Math.PI * radius * radius;
}
}
```

Working with Constructors

- In this example, the constructor is marked public.
- If this keyword is omitted, the constructor will be private (just like any other method and field).
- If the constructor is private, it cannot be used outside the class.

Overloading Constructors

- When more than one constructor with the same name is defined in the same class, they are called overloaded, while keeping parameters different for each constructor.
- We can overload constructors in different ways as follows:
  - By using different type of arguments.
  - By using different number of arguments.
  - By using different order of arguments.

Example:

```
Public Add (int a, float b);

Public Add (float a, int b);

Public Add ( string a, int b);
```

Partial Classes

- A class can contain a number of methods, fields, and constructors.
- A highly functional class can become quite large.
- With C#, you can split the source code for a class into separate files so that you can organize the definition of a large class into smaller pieces that are easier to manage.
- When you split a class across multiple files, you define the parts of the class by using the partial keyword in each file.
- For example, if the Circle class is split between two files called circ1.cs (containing the constructors)
- and circ2.cs (containing the methods and fields), the contents of circ1.cs
- look like this:

Partial Classes (Example)

```
partial class Circle
{
public Circle() // default constructor
{
this.radius = 0;
}
public Circle(int initialRadius) // overloaded constructor
{
this.radius = initialRadius; }
}

}
```

The contents of circ2.cs look like this:

```
partial class Circle
{
private int radius;
public double Area()
{
return Math.PI * this.radius * this.radius;
}
}
```

Static Classes

- A static class is declared with the help of static keyword.
- A static class can only contain static data members, static methods, and a static constructor.
- It is not allowed to create objects of the static class.
- Static classes are covered, means one cannot inherit a static class from another class.

Static Classes

- A static class is declared with the help of static keyword.
- A static class can only contain static data members, static methods, and a static constructor.
- It is not allowed to create objects of the static class.
- Static classes are covered, means one cannot inherit a static class from another class.

Anonymous Classes

- You create an anonymous class simply by using the new keyword and a pair of braces defining the fields and values that you want the class to contain, like this:

  e.g. myAnonymousObject = new { Name = "John ", Age = 47 };

- This class contains public fields called Name (initialized to the string “John“) and Age (initialized to the integer 47).

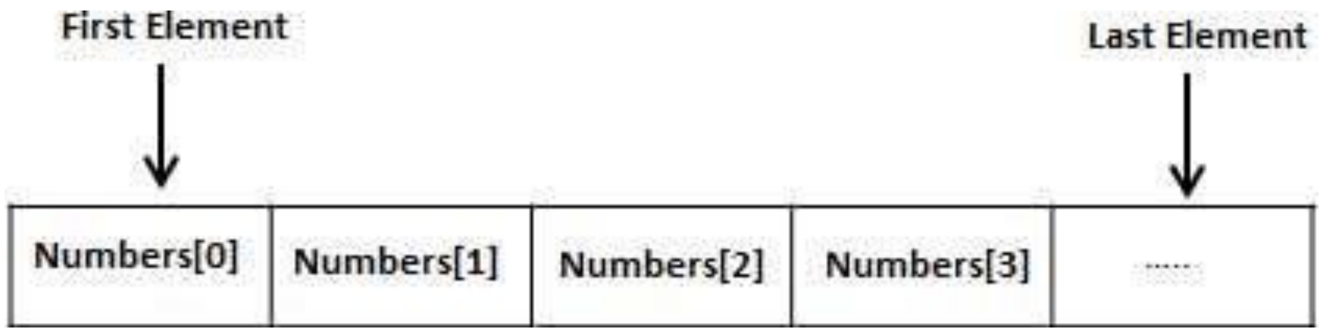
- The compiler infers the types of the fields from the types of the data you specify to initialize them.

Arrays

- An array stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but t is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Arrays

- A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

- To declare an array in C#
  - Datatype is used to specify the type of elements in the array.
  - [ ] specifies the rank of the array. The rank specifies the size of the array.
  - arrayName specifies the name of the array.
  - For example, double[ ] balance;

Declaring Arrays (Initializing)

- Declaring an array does not initialize the array in the memory.
- When the array variable is initialized, you can assign values to the array.
- Array is a reference type, so you need to use the new keyword to create an instance of the array.
- For example, double[] balance = new double[10];

Assigning Values to an Array (Initializing Methods)

1. You can assign values to individual array elements, by using the index number, like;

   - double[ ] balance = new double[10];
   - balance[0] = 4500.0;

1. double[ ] balance = { 2340.0, 4523.69, 3421.0};

2. int [ ] marks = new int[5] { 99, 98, 92, 97, 95};

3. You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location:

   - int [] marks = new int[] { 99, 98, 92, 97, 95};
   - int[ ] score = marks;

Accessing Array Elements

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.
  For example: double salary = balance[9];

```
using System;

namespace ArrayApplication {
   class MyArray {
      static void Main(string[] args) {
         int []  n = new int[10]; /* n is an array of 10 integers */
         int i,j;

         /* initialize elements of array n */
         for ( i = 0; i < 10; i++ ) {
            n[ i ] = i + 100;
         }

         /* output each array element's value */
         for (j = 0; j < 10; j++ ) {
            Console.WriteLine("Element[{0}] = {1}", j, n[j]);
         }
         Console.ReadKey();
      }
   }
}
```

Output:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Arrays (Some More Methods)

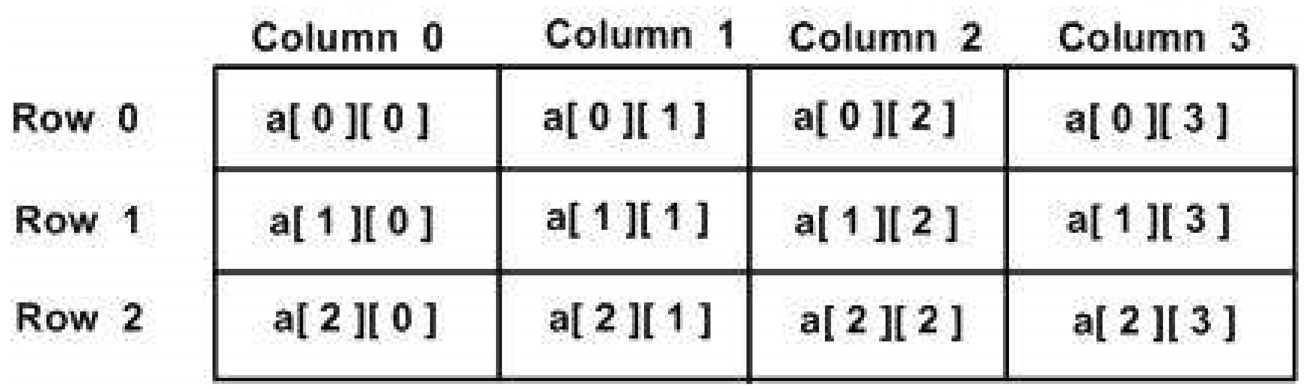
Following are the method which are mentioned below.

1. **Multi-dimensional arrays:** C# supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.

2. **Jagged arrays:** C# supports multidimensional arrays, which are arrays of arrays.

Two Dimensional Array

- The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is a list of one-dimensional arrays.
- A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns.
- Following is a 2-dimensional array, which contains 3 rows and 4 columns arrays.



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Accessing Two-Dimensional Arrays

- Multidimensional arrays may be initialized by specifying bracketed values for each row. The Following array is with 3 rows and each row has 4 columns.
- Initializing Two-Dimensional Arrays (Example)

```
int [,] a = new int [3,4] {
{0, 1, 2, 3} , /* initializers for row indexed by 0 */
{4, 5, 6, 7} , /* initializers for row indexed by 1 */
{8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

Initializing Two-Dimensional Arrays (Example)

- int val = [2, 3]

```
using System;

namespace ArrayApplication {
   class MyArray {
      static void Main(string[] args) {
         /* an array with 5 rows and 2 columns*/
         int[,] a = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };
         int i, j;

         /* output each array element's value */
         for (i = 0; i < 5; i++) {

            for (j = 0; j < 2; j++) {
               Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i,j]);
            }
         }
         Console.ReadKey();
      }
   }
}
```

Jagged Arrays

- In C#, ordinary multidimensional arrays are sometimes referred to as rectangular arrays. Each dimension has a regular shape.
- So, multidimensional arrays can consume a lot of memory.
- If the application uses only some of the data in each column, allocating memory for unused elements is a waste.
- In this scenario, you can use a jagged array, for which each column has a different length, like this:

```
int[][] items = new int[4][ ];
int[] columnForRow0 = new int[3];
int[] columnForRow1 = new int[10];
int[] columnForRow2 = new int[40];
int[] columnForRow3 = new int[25];
```

String Handling

- In C#, string is an object of System.String class that represent sequence of characters. We can perform many operations on strings such as concatenation, comparison, getting substring, search, trim, replacement etc.
- string vs String
- **string** s1 = "hello";//creating string using string keyword
- String s2 = "welcome";//creating string using String class

String Handling (Example)

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "hello";

        char[] ch = { 'c', 's', 'h', 'a', 'r', 'p' };
        string s2 = new string(ch);

        Console.WriteLine(s1);
        Console.WriteLine(s2);
    }
}
```

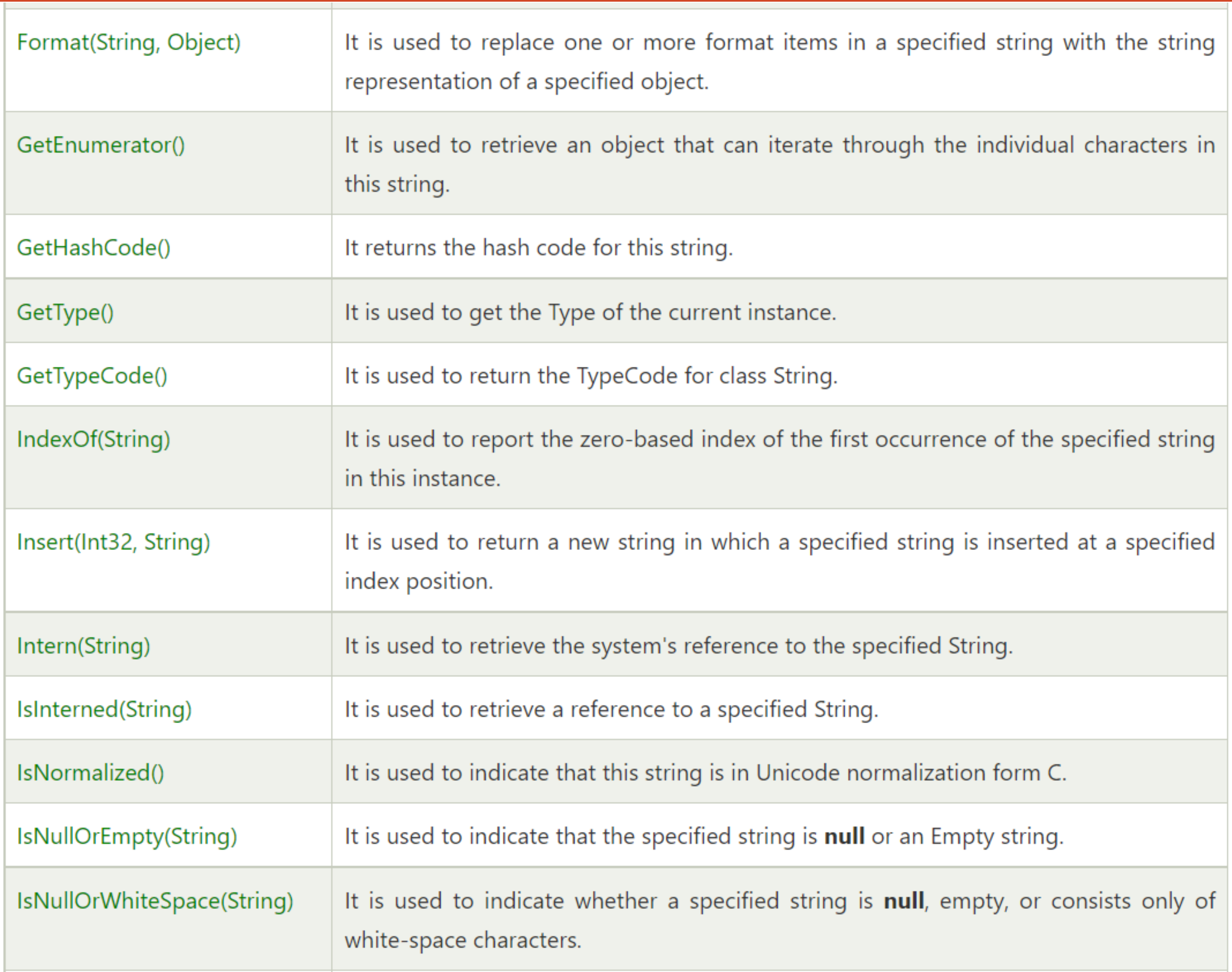
Output:

```
hello
csharp
```

String Handling (Example)

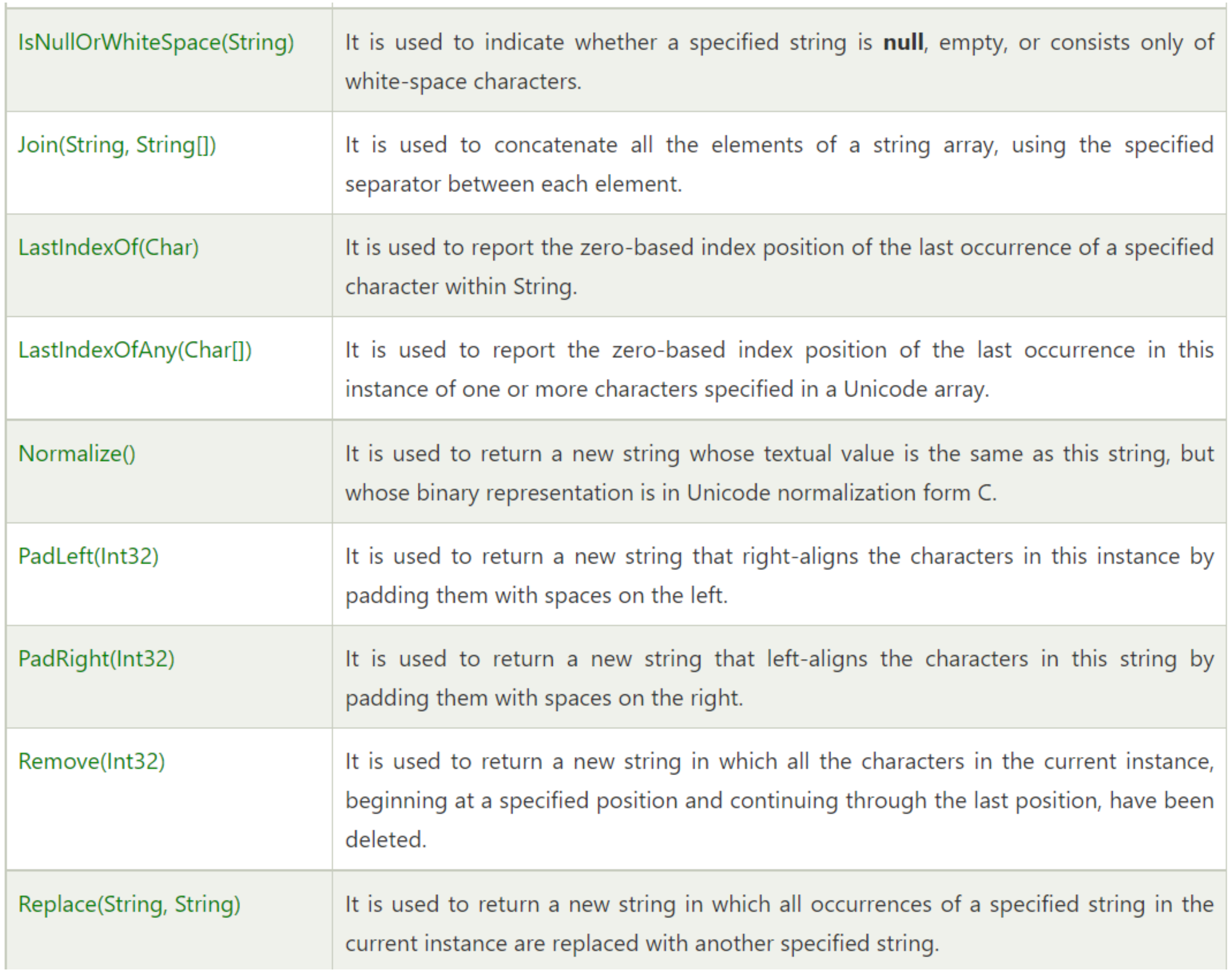
Clone()	It is used to return a reference to this instance of String.
Compare(String, String)	It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.
CompareOrdinal(String, String)	It is used to compare two specified String objects by evaluating the numeric values of the corresponding Char objects in each string..
CompareTo(String)	It is used to compare this instance with a specified String object. It indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.
Concat(String, String)	It is used to concatenate two specified instances of String.
Contains(String)	It is used to return a value indicating whether a specified substring occurs within this string.
Copy(String)	It is used to create a new instance of String with the same value as a specified String.
CopyTo(Int32, Char[], Int32, Int32)	It is used to copy a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.
EndsWith(String)	It is used to check that the end of this string instance matches the specified string.
Equals(String, String)	It is used to determine that two specified String objects have the same value.

String Handling (Example)



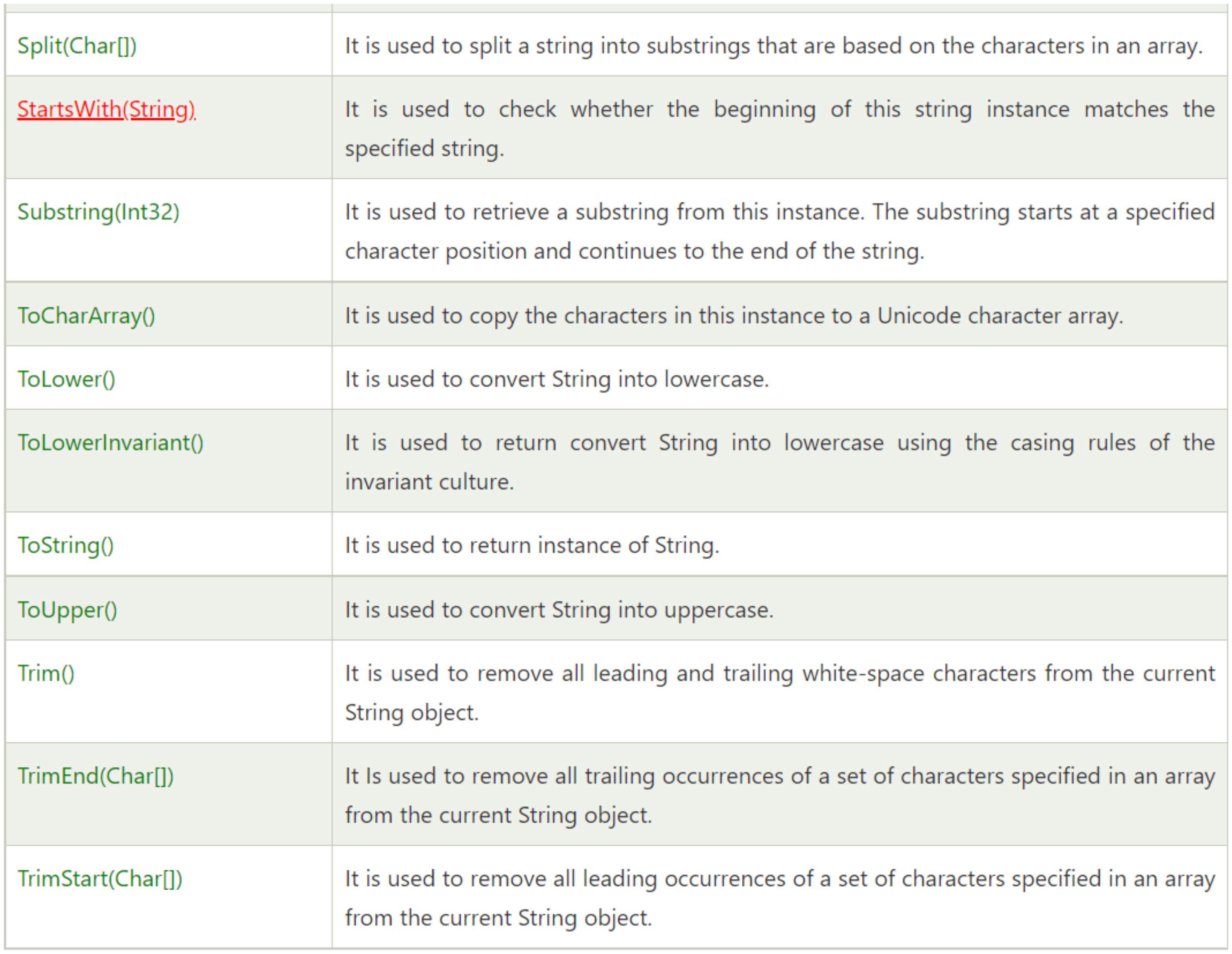
Format(String, Object)	It is used to replace one or more format items in a specified string with the string representation of a specified object.
GetEnumerator()	It is used to retrieve an object that can iterate through the individual characters in this string.
GetHashCode()	It returns the hash code for this string.
GetType()	It is used to get the Type of the current instance.
GetTypeCode()	It is used to return the TypeCode for class String.
IndexOf(String)	It is used to report the zero-based index of the first occurrence of the specified string in this instance.
Insert(Int32, String)	It is used to return a new string in which a specified string is inserted at a specified index position.
Intern(String)	It is used to retrieve the system's reference to the specified String.
IsInterned(String)	It is used to retrieve a reference to a specified String.
IsNormalized()	It is used to indicate that this string is in Unicode normalization form C.
IsNullOrEmpty(String)	It is used to indicate that the specified string is **null** or an Empty string.
IsNullOrWhiteSpace(String)	It is used to indicate whether a specified string is **null**, empty, or consists only of white-space characters.

String Handling (Example)



IsNullOrWhiteSpace(String)	It is used to indicate whether a specified string is **null**, empty, or consists only of white-space characters.
Join(String, String[])	It is used to concatenate all the elements of a string array, using the specified separator between each element.
LastIndexOf(Char)	It is used to report the zero-based index position of the last occurrence of a specified character within String.
LastIndexOfAny(Char[])	It is used to report the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array.
Normalize()	It is used to return a new string whose textual value is the same as this string, but whose binary representation is in Unicode normalization form C.
PadLeft(Int32)	It is used to return a new string that right-aligns the characters in this instance by padding them with spaces on the left.
PadRight(Int32)	It is used to return a new string that left-aligns the characters in this string by padding them with spaces on the right.
Remove(Int32)	It is used to return a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.
Replace(String, String)	It is used to return a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.

String Handling (Example)



Split(Char[])	It is used to split a string into substrings that are based on the characters in an array.
StartsWith(String)	It is used to check whether the beginning of this string instance matches the specified string.
Substring(Int32)	It is used to retrieve a substring from this instance. The substring starts at a specified character position and continues to the end of the string.
ToCharArray()	It is used to copy the characters in this instance to a Unicode character array.
ToLower()	It is used to convert String into lowercase.
ToLowerInvariant()	It is used to return convert String into lowercase using the casing rules of the invariant culture.
ToString()	It is used to return instance of String.
ToUpper()	It is used to convert String into uppercase.
Trim()	It is used to remove all leading and trailing white-space characters from the current String object.
TrimEnd(Char[])	It Is used to remove all trailing occurrences of a set of characters specified in an array from the current String object.
TrimStart(Char[])	It is used to remove all leading occurrences of a set of characters specified in an array from the current String object.