IT FACULTY

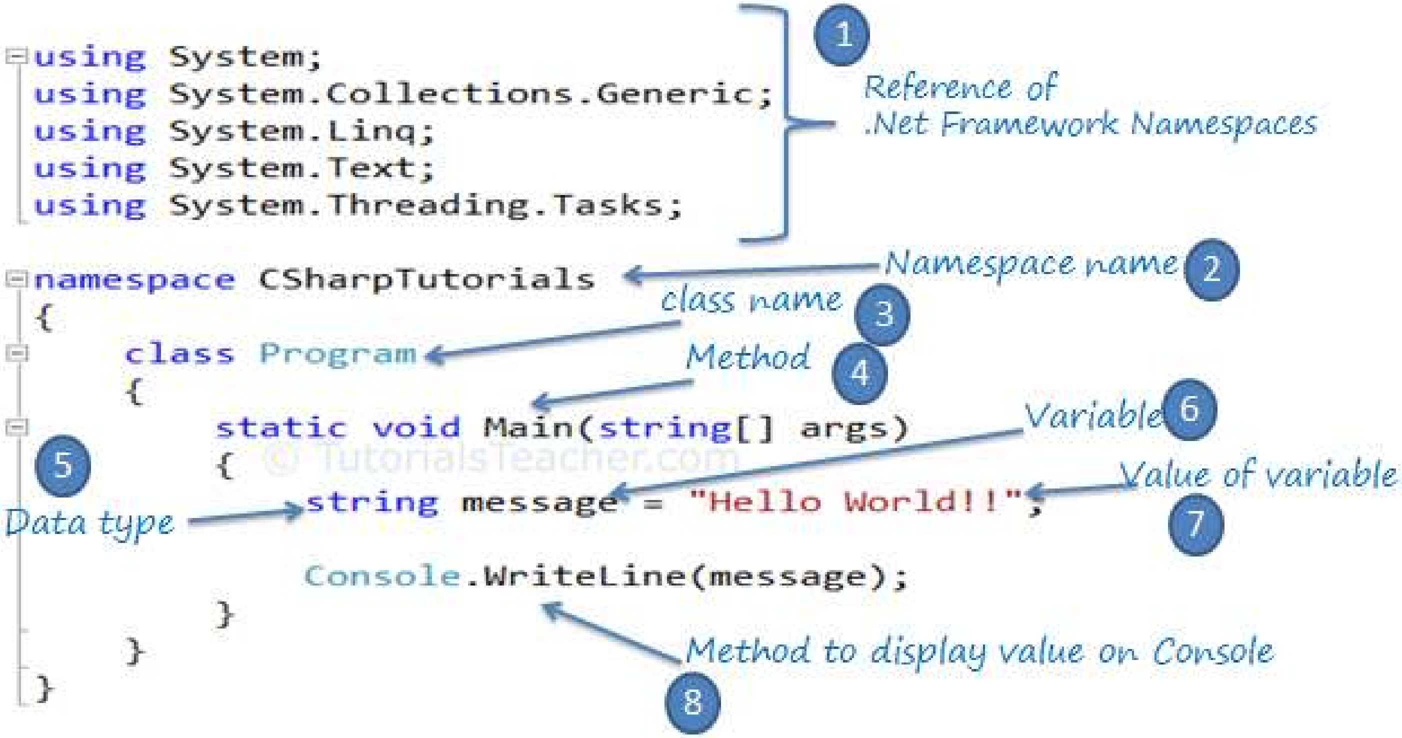
C# FUNDAMENTAL

BUILDING C# APPLICATIONS

- Using csc.exe:
  - IDE: Visual Studio Code (VS Code), Notepad;
  - Configuration:
    - Command-line: use ***csc.exe***;
    - Config file ***task.json***.
- Using Visual Studio: IDE Visual Studio
  - Code Definition View;
  - Code Refactoring;
  - Visual Class Designer;
  - Object Test Bench.

STRUCTURE OF A C# PROGRAM

- Console App: C# code Structure:
  - ***using***: take the reference;
  - Method ***Main()***: entry point of the Console App.



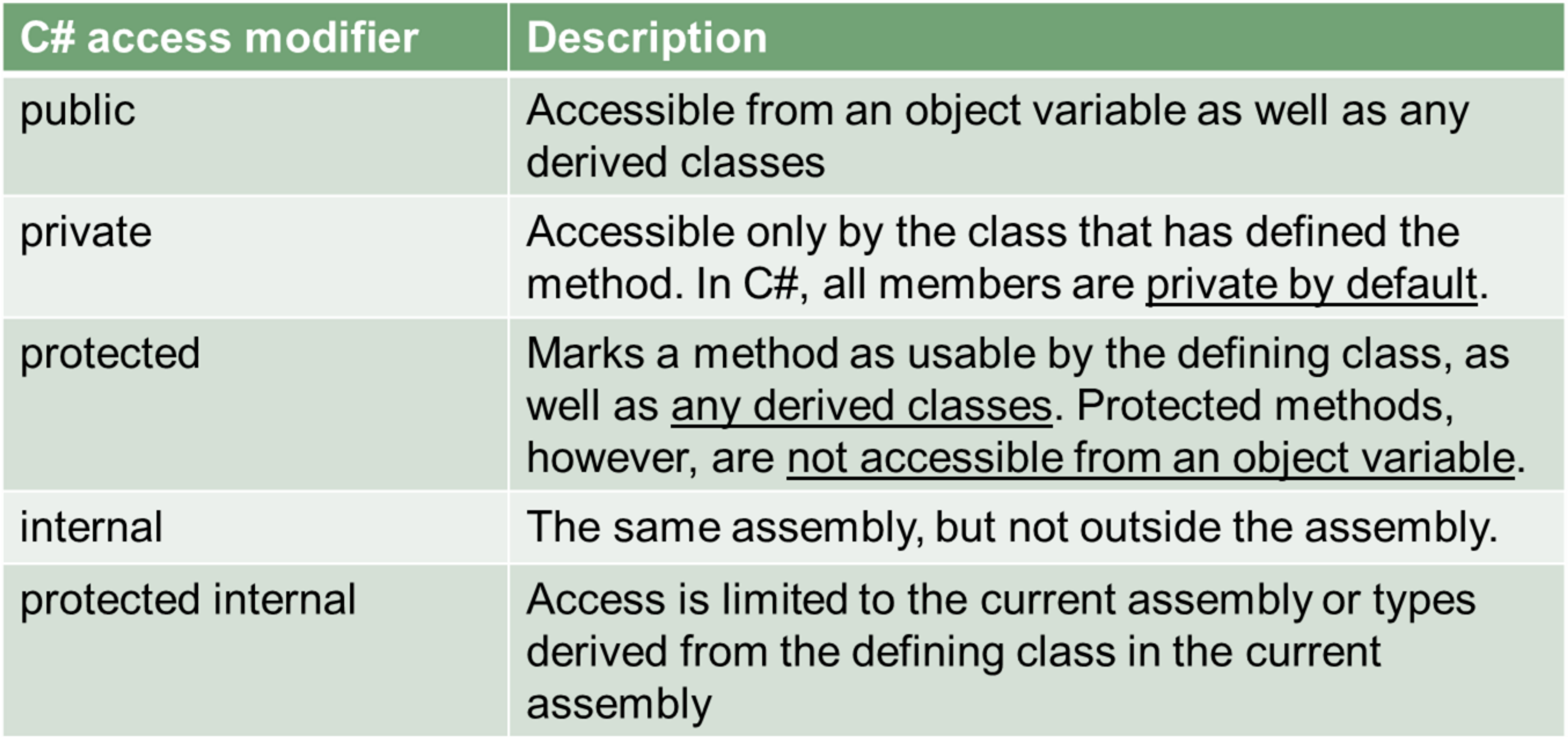
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CSharpTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
            string message = "Hello World!!";

            Console.WriteLine(message);
        }
    }
}
```

1
Reference of
.Net Framework Namespaces
Namespace name
2
class name
3
Method
4
5
Data type
Variable
6
Value of variable
7
Method to display value on Console
8
© TutorialsTeacher.com

STRUCTURE OF A C# PROGRAM

- Project:
  - **AssemblyInfo.cs**;
  - **References**;
  - **App.Config (Web.Config)**;
  - **Program.cs**: *static void Main(string[] args)*.

MEMBER VISIBILITY



C# access modifier	Description
public	Accessible from an object variable as well as any derived classes
private	Accessible only by the class that has defined the method. In C#, all members are private by default.
protected	Marks a method as usable by the defining class, as well as any derived classes. Protected methods, however, are not accessible from an object variable.
internal	The same assembly, but not outside the assembly.
protected internal	Access is limited to the current assembly or types derived from the defining class in the current assembly

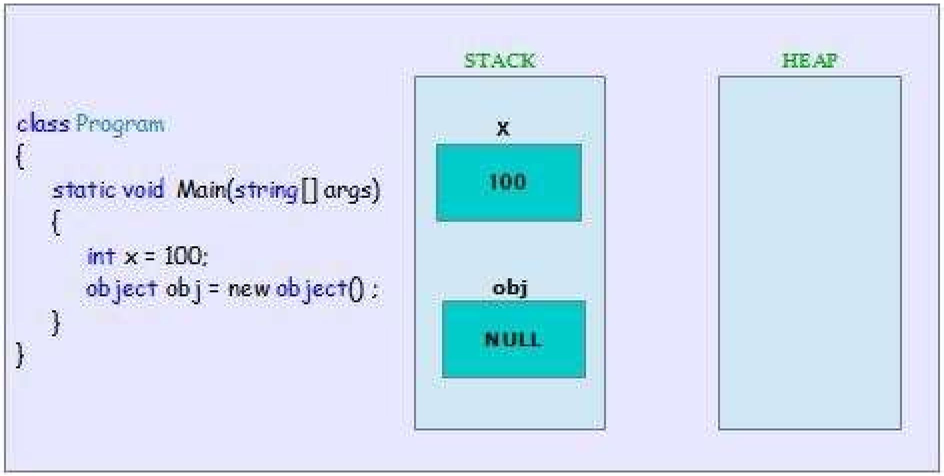
- Types (classes, interfaces, structures, enumerations, and delegates) are limited to **public** or **internal** (default).

I/O

- Class **Console**:
  - Namespace: System
  - Assemblies: System.Console.dll, mscorlib.dll, netstandard.dll
  - Represents the standard input, output, and error streams for console applications. This class cannot be inherited.
  - Static method:
    - Write();
    - WriteLine();
    - Read();
    - ReadLine();
    - ReadKey();

VALUE TYPES & REFERENCE TYPES

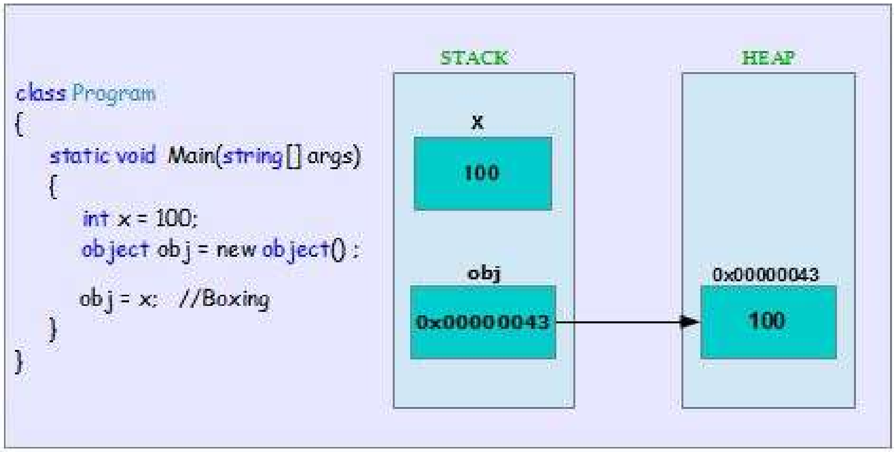
- ***Value types***, which include all numerical data types (int, float, etc.), as well as enumerations and structures, are allocated on the stack:
  - Value types can be quickly removed from memory once they fall out of the defining scope.
- In contrast, ***reference types*** (classes) are allocated on the managed heap:
  - These objects stay in memory until the .NET garbage collector destroys them.



```
class Program
{
    static void  Main(string[] args)
    {
        int x = 100;
        object obj = new object() ;
    }
}
```

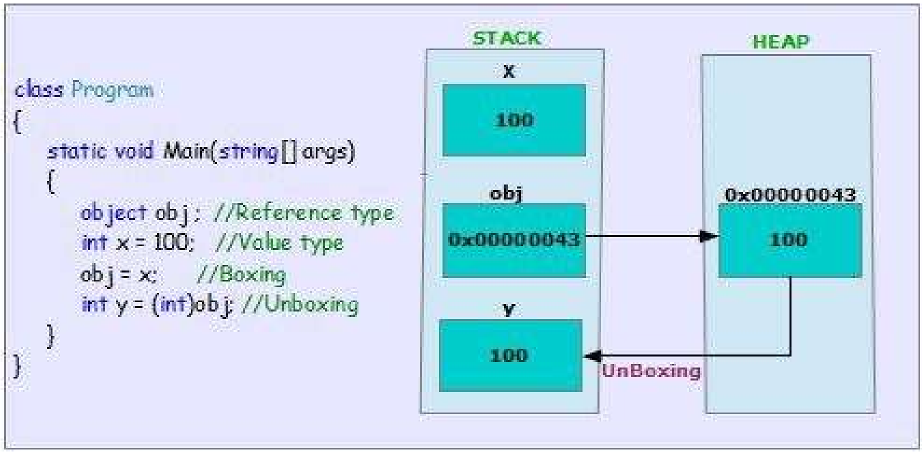
BOXING AND UNBOXING OPERATIONS

- ***Boxing***: converting a value type into a corresponding reference type by storing the variable in a System.Object.
- ***Unboxing***: converting the value held in the object reference back into a corresponding value type.



```
class Program
{
    static void Main(string[] args)
    {
        int x = 100;
        object obj = new object() ;

        obj = x;   //Boxing
    }
}
```

STACK
HEAP
x
100
obj
0x00000043
0x00000043
100



```
class Program
{
    static void Main(string[] args)
    {
        object obj ; //Reference type
        int x = 100;  //Value type
        obj = x;     //Boxing
        int y = (int)obj; //Unboxing
    }
}
```

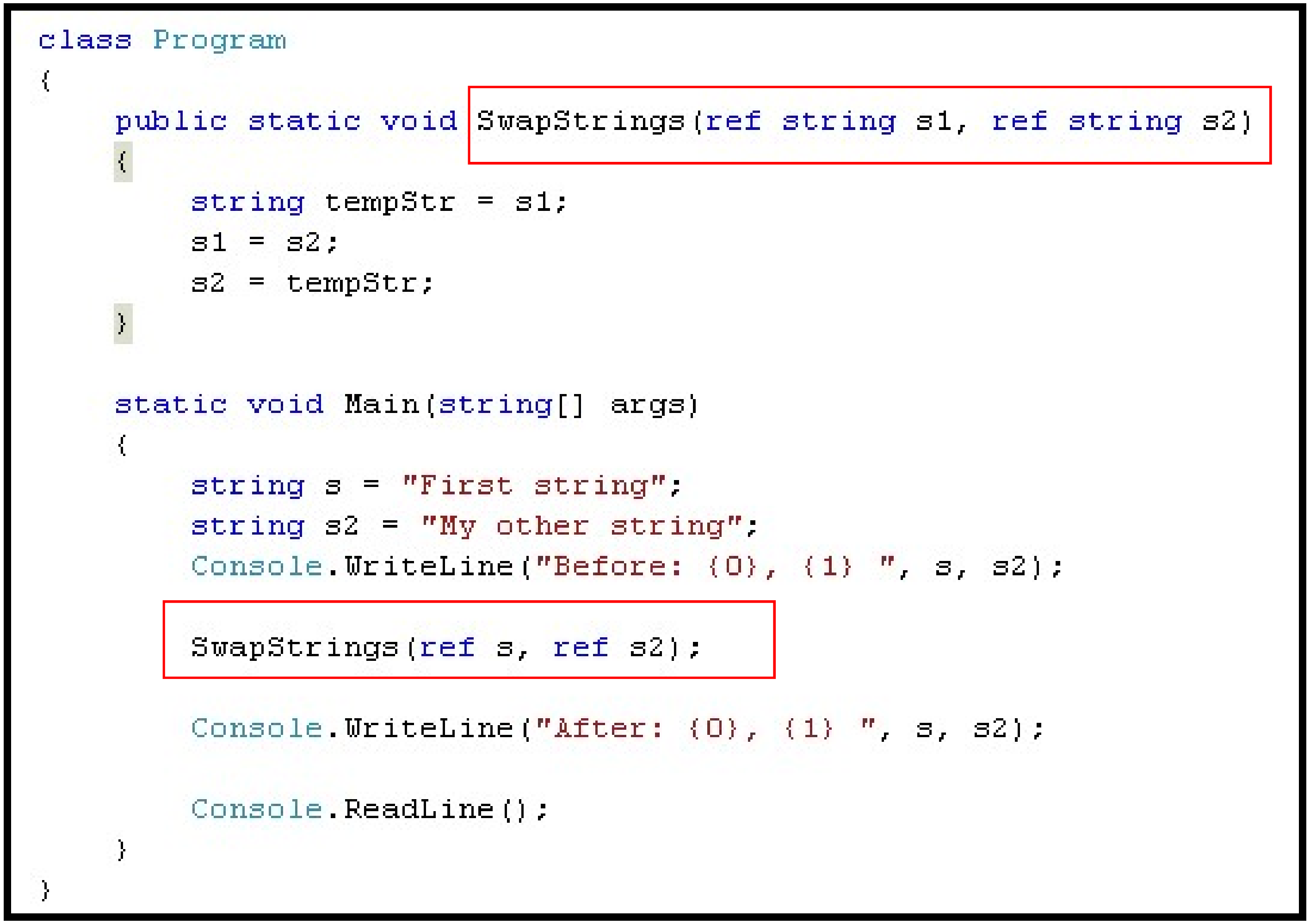
STACK
HEAP
x
100
obj
0x00000043
0x00000043
100
y
100
UnBoxing

METHOD PARAMETER MODIFIERS

Parameter Modifier	**Description**
(none)	Parameters are passed by value
out	Output parameters are assigned by the method being called (and therefore *passed by reference*).
params	Allows having a number of identically typed arguments as a single logical parameter. A method can have *only a single params* modifier, and it must be *the final* parameter of the method.
ref	Parameters are passed by reference.

METHOD PARAMETER MODIFIERS

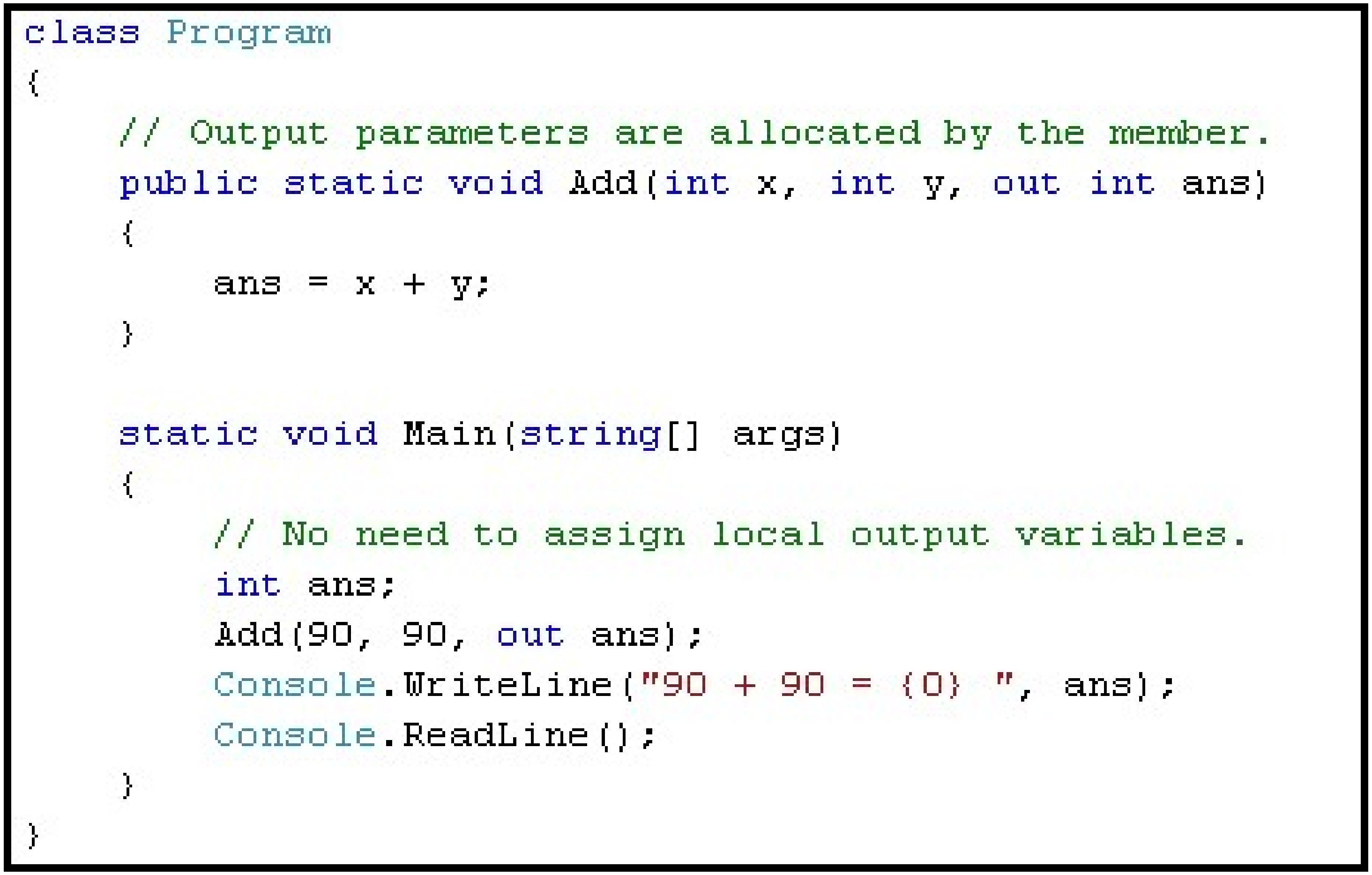
- ***ref*:**



```
class Program
{
    public static void SwapStrings(ref string s1, ref string s2)
    {
        string tempStr = s1;
        s1 = s2;
        s2 = tempStr;
    }

    static void Main(string[] args)
    {
        string s = "First string";
        string s2 = "My other string";
        Console.WriteLine("Before: {0}, {1} ", s, s2);

        SwapStrings(ref s, ref s2);

        Console.WriteLine("After: {0}, {1} ", s, s2);

        Console.ReadLine();
    }
}
```

METHOD PARAMETER MODIFIERS

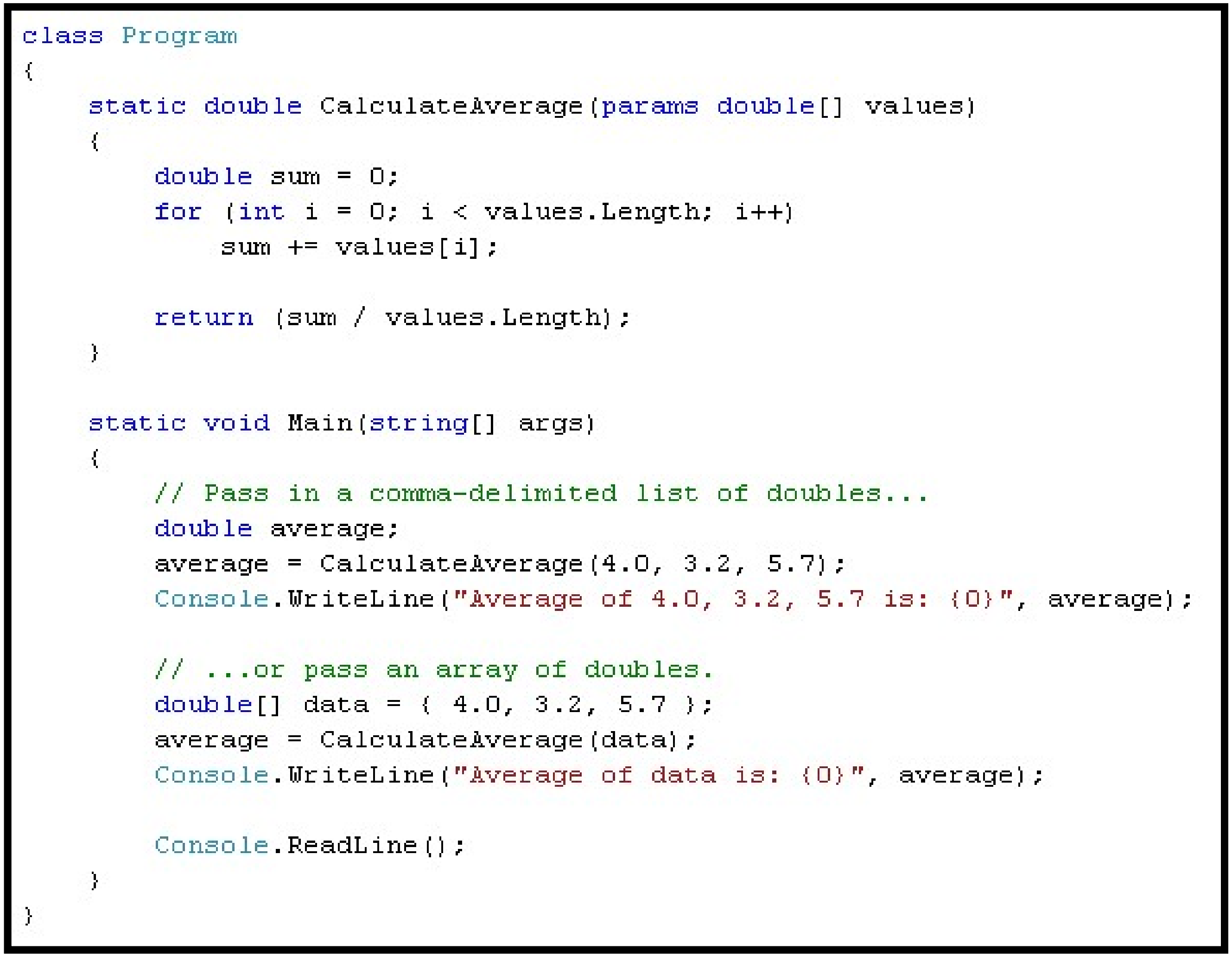
- *out*:



```
class Program
{
    // Output parameters are allocated by the member.
    public static void Add(int x, int y, out int ans)
    {
        ans = x + y;
    }

    static void Main(string[] args)
    {
        // No need to assign local output variables.
        int ans;
        Add(90, 90, out ans);
        Console.WriteLine("90 + 90 = {0} ", ans);
        Console.ReadLine();
    }
}
```

METHOD PARAMETER MODIFIERS

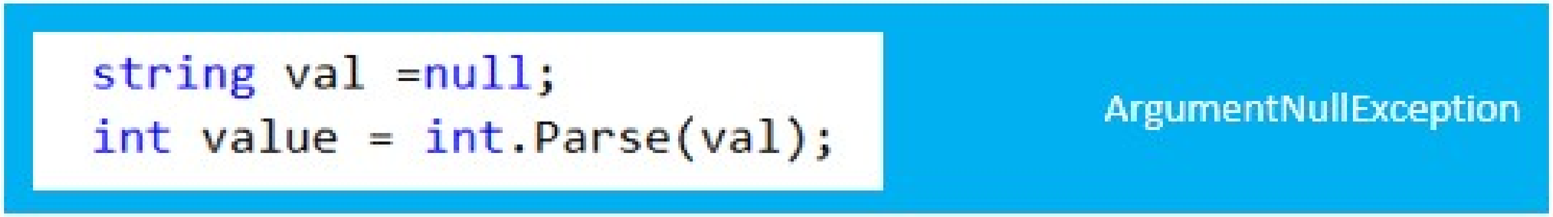
- ***param*:**



```
class Program
{
    static double CalculateAverage(params double[] values)
    {
        double sum = 0;
        for (int i = 0; i < values.Length; i++)
            sum += values[i];

        return (sum / values.Length);
    }

    static void Main(string[] args)
    {
        // Pass in a comma-delimited list of doubles...
        double average;
        average = CalculateAverage(4.0, 3.2, 5.7);
        Console.WriteLine("Average of 4.0, 3.2, 5.7 is: {0}", average);

        // ...or pass an array of doubles.
        double[] data = { 4.0, 3.2, 5.7 };
        average = CalculateAverage(data);
        Console.WriteLine("Average of data is: {0}", average);

        Console.ReadLine();
    }
}
```

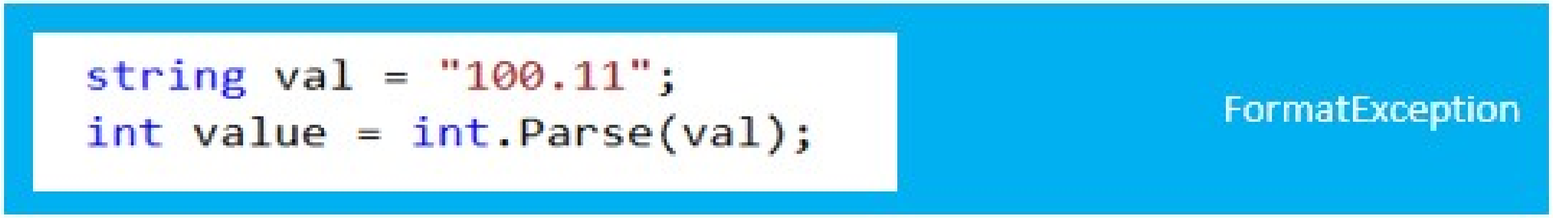
PARSE()

- ***Parse***:



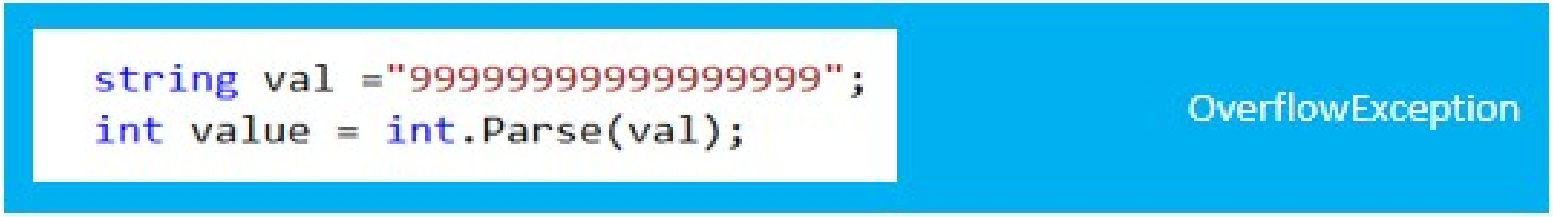
```
string val =null;
int value = int.Parse(val);
```

ArgumentNullException



```
string val = "100.11";
int value = int.Parse(val);
```

FormatException

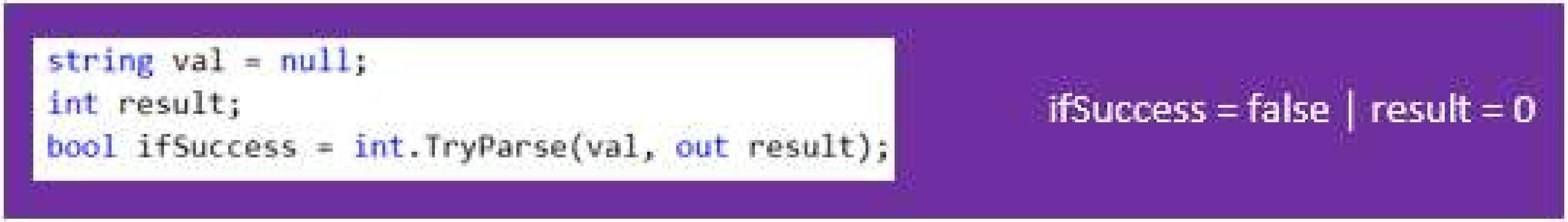


```
string val ="9999999999999999";
int value = int.Parse(val);
```

OverflowException

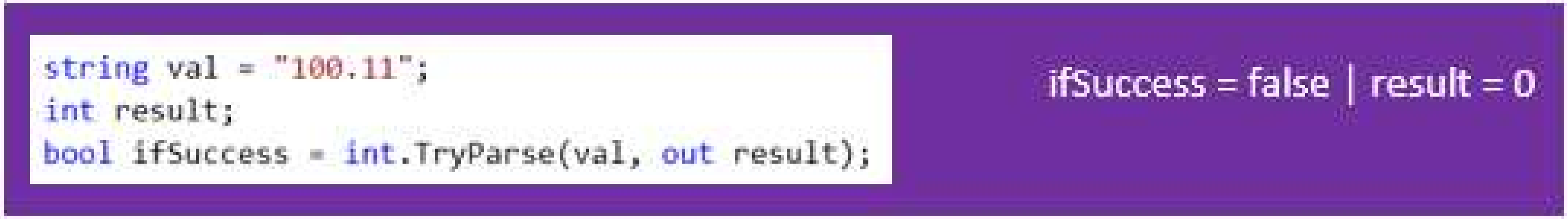
TRYPARSE()

- *TryParse*:



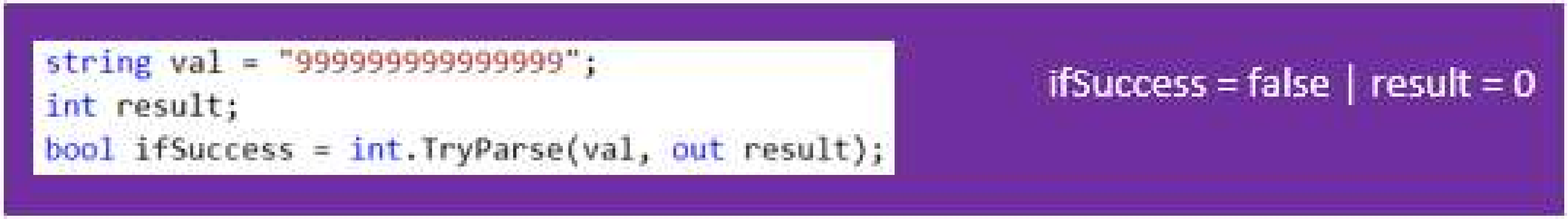
```
string val = null;
int result;
bool ifSuccess = int.TryParse(val, out result);
```

ifSuccess = false | result = 0



```
string val = "100.11";
int result;
bool ifSuccess = int.TryParse(val, out result);
```

ifSuccess = false | result = 0



```
string val = "999999999999999";
int result;
bool ifSuccess = int.TryParse(val, out result);
```

ifSuccess = false | result = 0

CLASS CONVERT

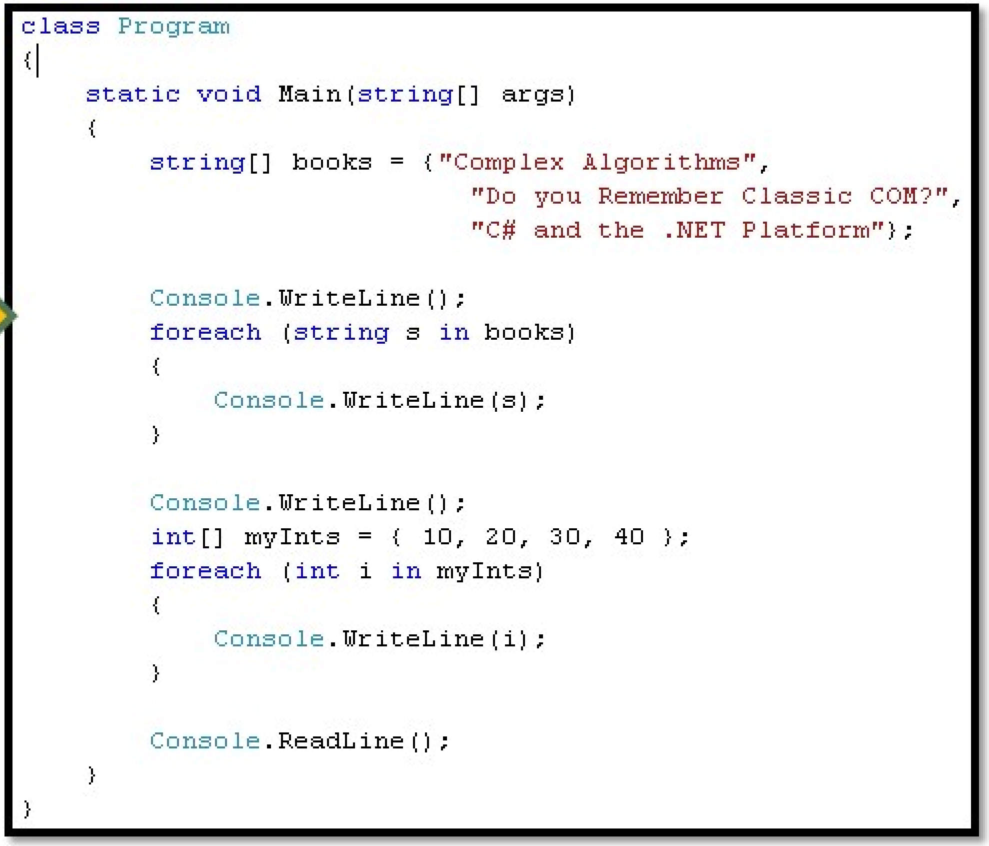
- More than one way to convert from one base type to another:
  - Namespace: ***System***, Class: ***Convert***, Methods: ***Static***.
  - Convert.ToDouble();
  - Convert.ToInt32();
  - Convert.ToString();
  - Convert.ToChar();
  - …

CONSTANT & READONLY

- ***Defining Constant Data***:
  - C# offers the ***const*** keyword to define variables with a fixed, unalterable value;
  - It is important to understand that the value assigned to a constant variable must be known at *compile time* → constant member cannot be assigned to an object reference;
  - All constant fields are *implicitly static*.
- ***Defining Read-Only Fields***:
  - Read-only fields allow you to establish a point of data whose value is not known at compile time, but that *should never change once established*;
  - Read-only fields *are not implicitly static*.

ITERATION CONSTRUCTS

- for loop
- **foreach/in loop**
- while loop
- do/while loop

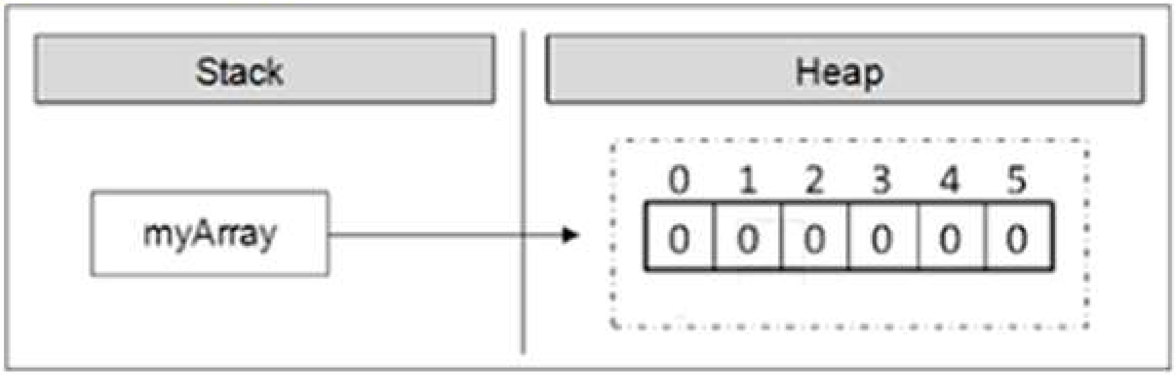


```
class Program
{
    static void Main(string[] args)
    {
        string[] books = {"Complex Algorithms",
                            "Do you Remember Classic COM?",
                            "C# and the .NET Platform"};

        Console.WriteLine();
        foreach (string s in books)
        {
            Console.WriteLine(s);
        }

        Console.WriteLine();
        int[] myInts = { 10, 20, 30, 40 };
        foreach (int i in myInts)
        {
            Console.WriteLine(i);
        }

        Console.ReadLine();
    }
}
```

SIMPLE ARRAY

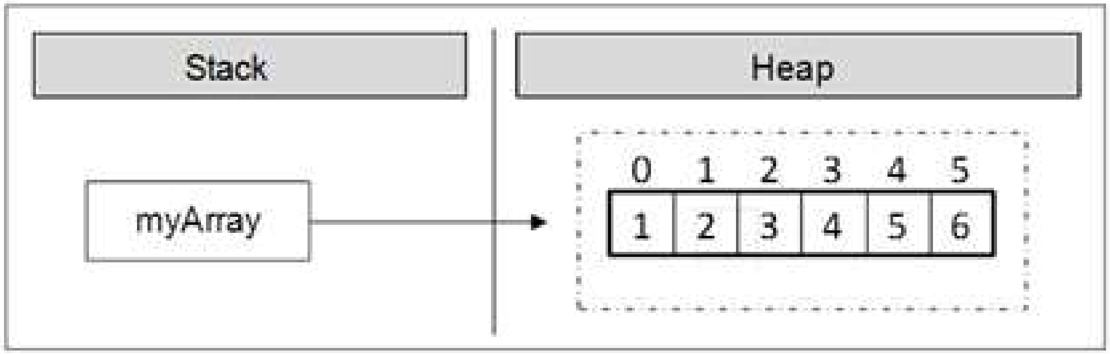
- Creation of an Array – the Operator **new**:

```
int[] myArray = new int[6];
```


Stack
Heap
myArray
0 1 2 3 4 5
0 0 0 0 0 0

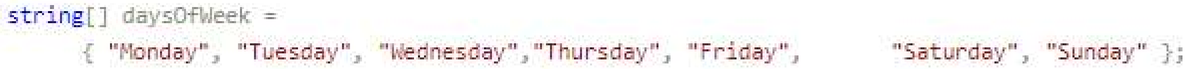
- Array Initialization and Default Values:

```
int[] myArray = { 1, 2, 3, 4, 5, 6 };
```

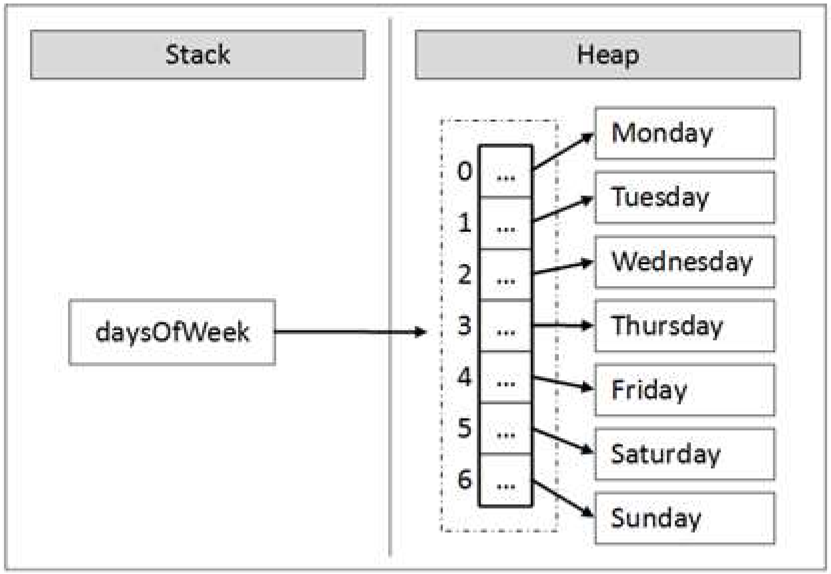

Stack
Heap
myArray
0 1 2 3 4 5
1 2 3 4 5 6

SIMPLE ARRAY

- Array Initialization and Default Values:

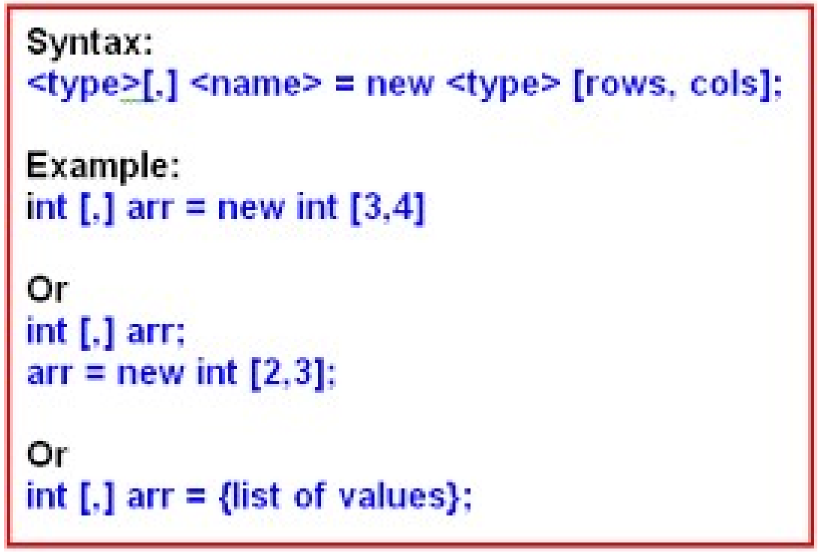


```
string[] daysOfWeek =
     { "Monday", "Tuesday", "Wednesday","Thursday", "Friday",      "Saturday", "Sunday" };
```


Stack
Heap
daysOfWeek
0
1
2
3
4
5
6
...
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

MULTI-DIMENSIONAL ARRAY

- Rectangular:



Syntax:

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

Example:

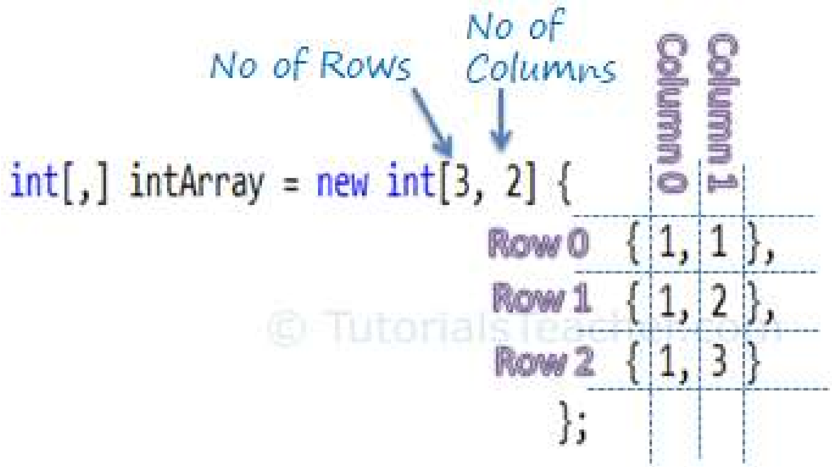
```
int [,] arr = new int [3,4]
```

Or

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

Or

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

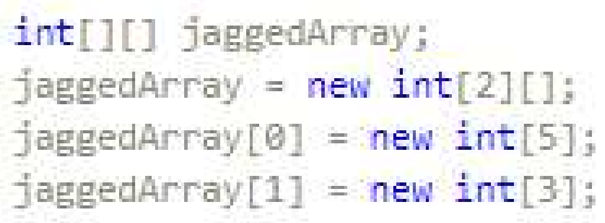

No of Rows
No of Columns
Column 0
Column 1
Row 0
Row 1
Row 2

```
int[,] intArray = new int[3, 2] {
                                  { 1, 1 },
                                  { 1, 2 },
                                  { 1, 3 }
                                };
```

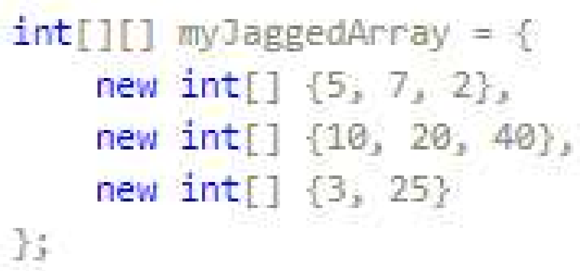
© TutorialsTeac

MULTI-DIMENSIONAL ARRAY

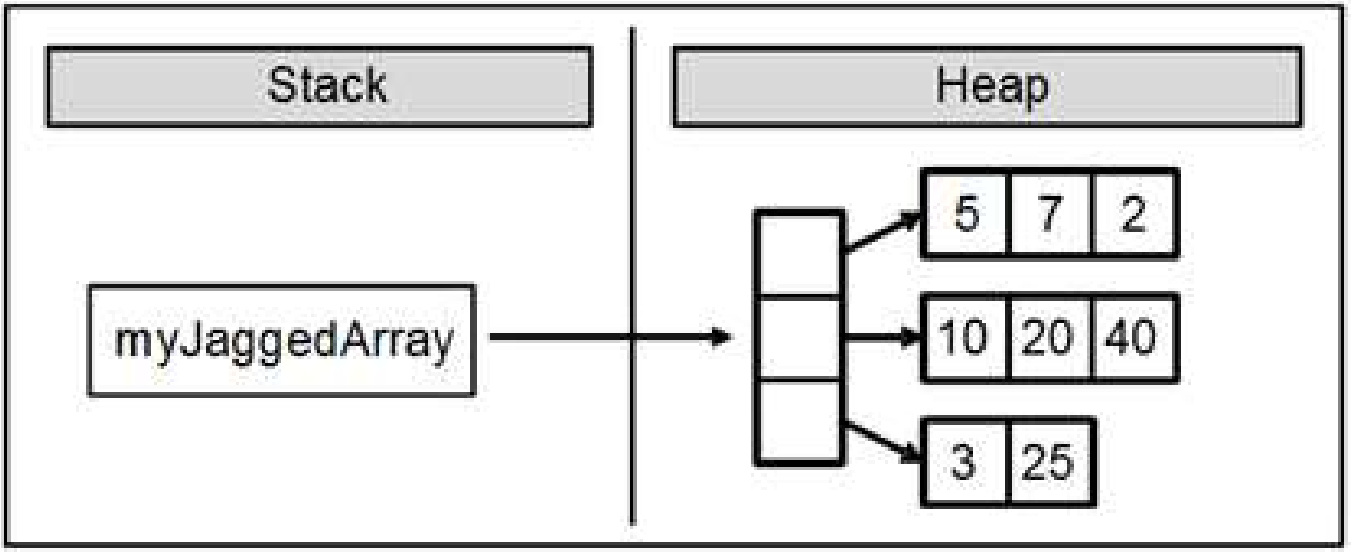
- A ***jagged array*** is an array of arrays.
- Declaration and Allocation:



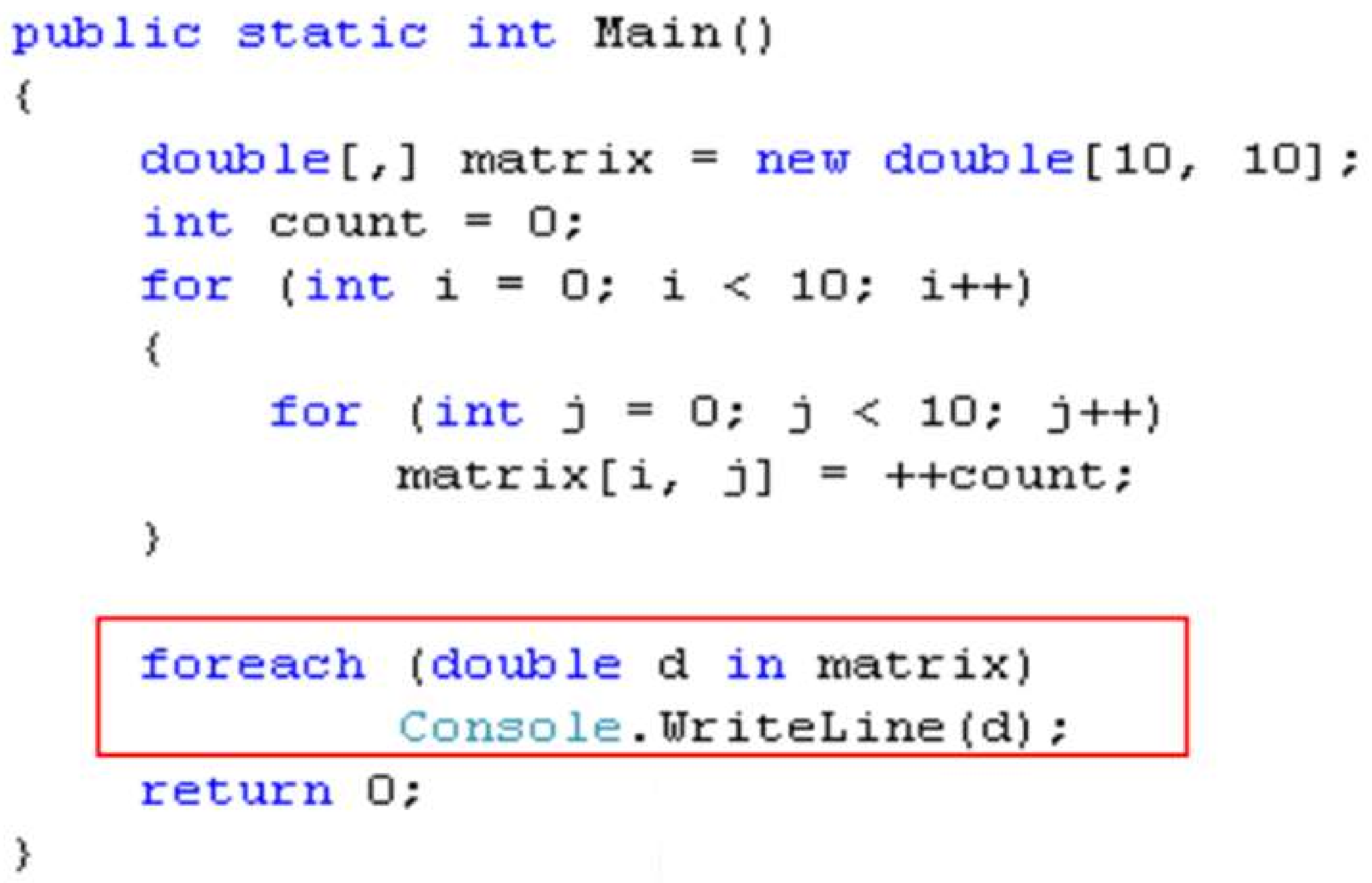
```
int[][] jaggedArray;
jaggedArray = new int[2][];
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[3];
```



```
int[][] myJaggedArray = {
     new int[] {5, 7, 2},
     new int[] {10, 20, 40},
     new int[] {3, 25}
};
```

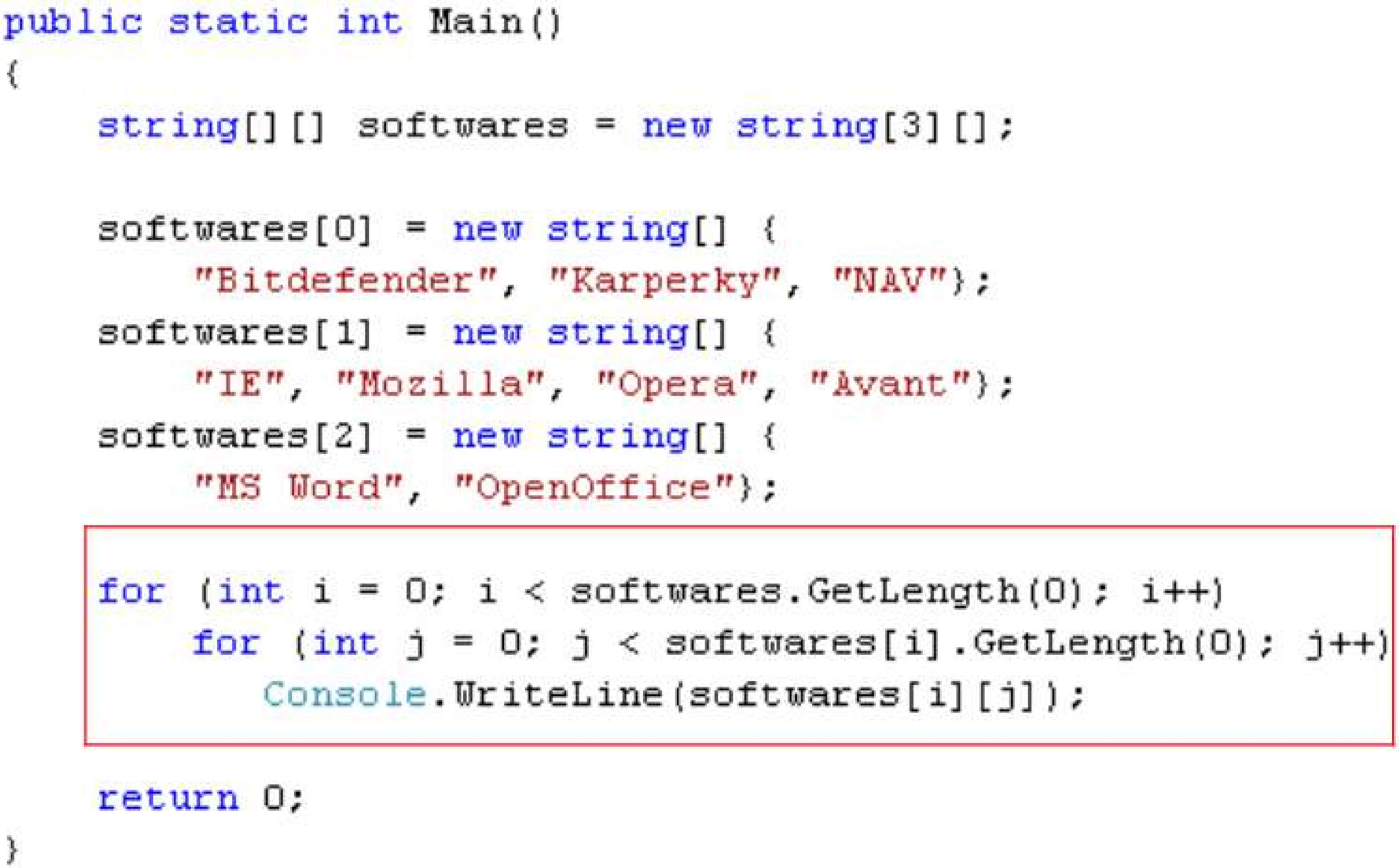

Stack
Heap
myJaggedArray
5
7
2
10
20
40
3
25

FOREACH WITH ARRAY



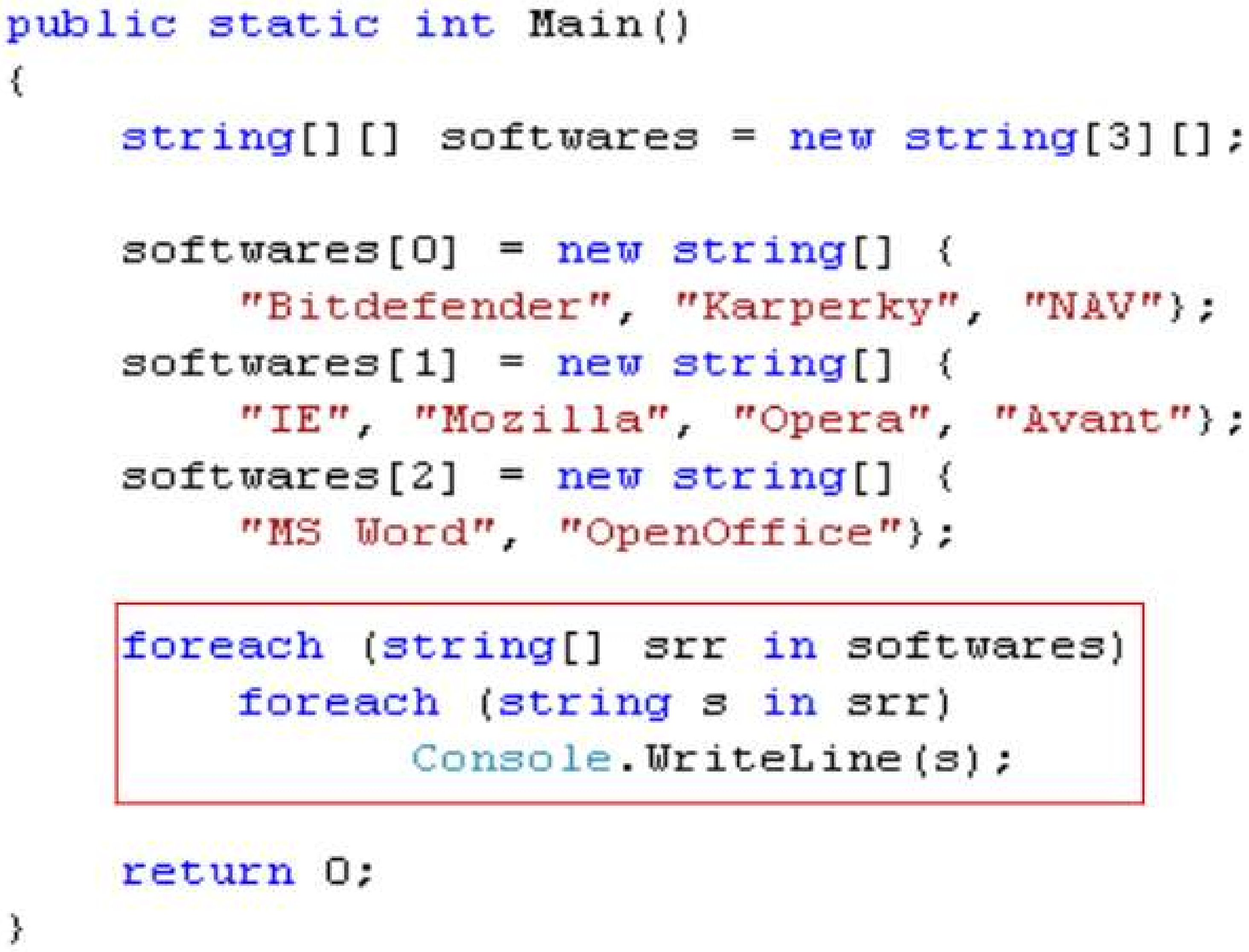
```
public static int Main()
{
    double[,] matrix = new double[10, 10];
    int count = 0;
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
            matrix[i, j] = ++count;
    }

    foreach (double d in matrix)
            Console.WriteLine(d);
    return 0;
}
```

FOREACH WITH ARRAY



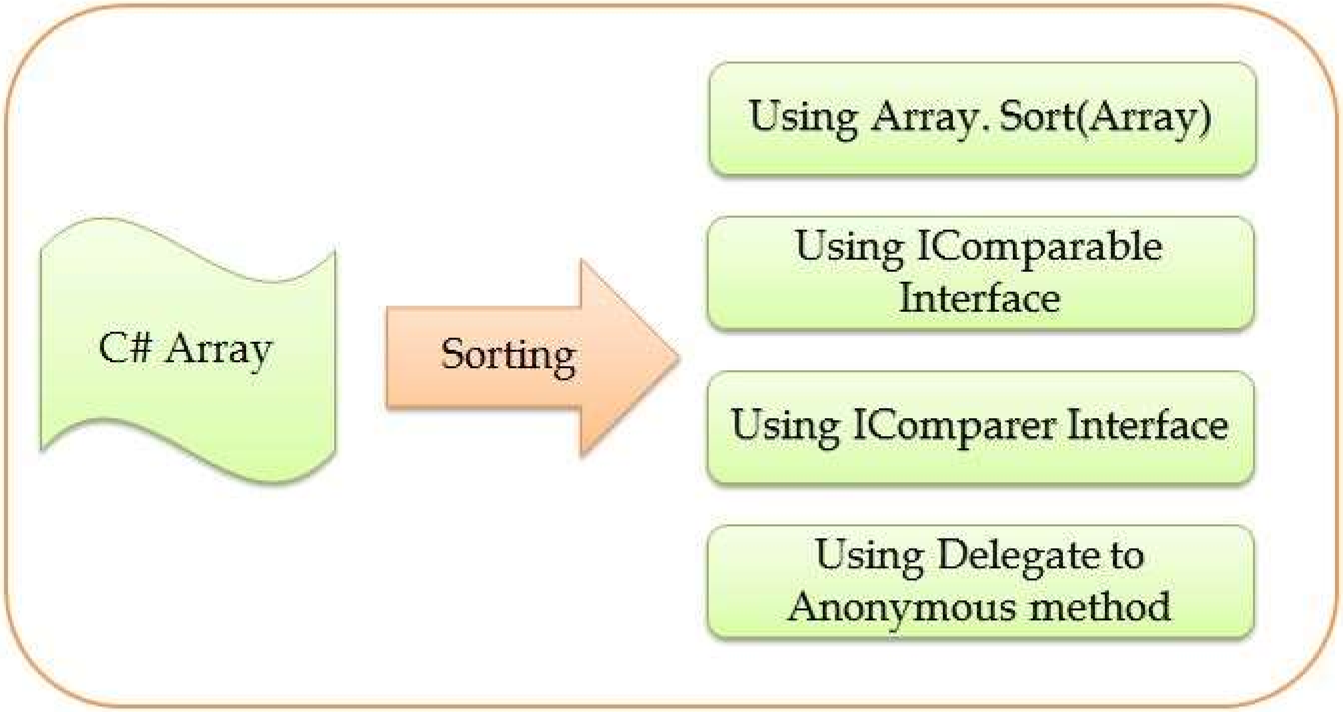
```
public static int Main()
{
    string[][] softwares = new string[3][];

    softwares[0] = new string[] {
        "Bitdefender", "Karperky", "NAV"};
    softwares[1] = new string[] {
        "IE", "Mozilla", "Opera", "Avant"};
    softwares[2] = new string[] {
        "MS Word", "OpenOffice"};

    for (int i = 0; i < softwares.GetLength(0); i++)
        for (int j = 0; j < softwares[i].GetLength(0); j++)
            Console.WriteLine(softwares[i][j]);

    return 0;
}
```

FOREACH WITH ARRAY



```
public static int Main()
{
    string[][] softwares = new string[3][];

    softwares[0] = new string[] {
        "Bitdefender", "Karperky", "NAV"};
    softwares[1] = new string[] {
        "IE", "Mozilla", "Opera", "Avant"};
    softwares[2] = new string[] {
        "MS Word", "OpenOffice"};

    foreach (string[] srr in softwares)
        foreach (string s in srr)
            Console.WriteLine(s);

    return 0;
}
```

WAY TO SORT AN ARRAY



C# Array

Sorting

- Using Array. Sort(Array)
- Using IComparable Interface
- Using IComparer Interface
- Using Delegate to Anonymous method

Thank You !

IT FACULTY