Object-oriented Programming with C#

Tahaluf Training Center 2021

Day 8

1. String Vs string C#
2. C# String Interpolation
3. Lambda Expressions
4. Constructors and Destructors
5. IDisposable Interface

String Vs string C#

One of the questions that many novice C# programmers ask is: “What is the difference between string and String?”

String Vs string C#

- ✓ String (capital S) is a class in the .NET framework in the System namespace. The fully qualified name is System.String.

- ✓ Whereas, the lower case string is an alias of System.String.

So, technically there is no difference between string and String, but it is common practice to declare a variable using C# keywords.

String Vs string C#

- ✓ The only tiny difference is that if you use the **String** class, you need to import the **System** namespace on top of your file, whereas you don't have to do this when using the **string** keyword.

String Vs string C#

- ✓ It is recommended to use string (lower case) over String. However, it's a matter of choice. You can use any of them.
- ✓ Many developers use string to declare variables in C# and use System.String class to use any built-in string methods e.g., String.IsNullOrEmpty().

String Vs string C#

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

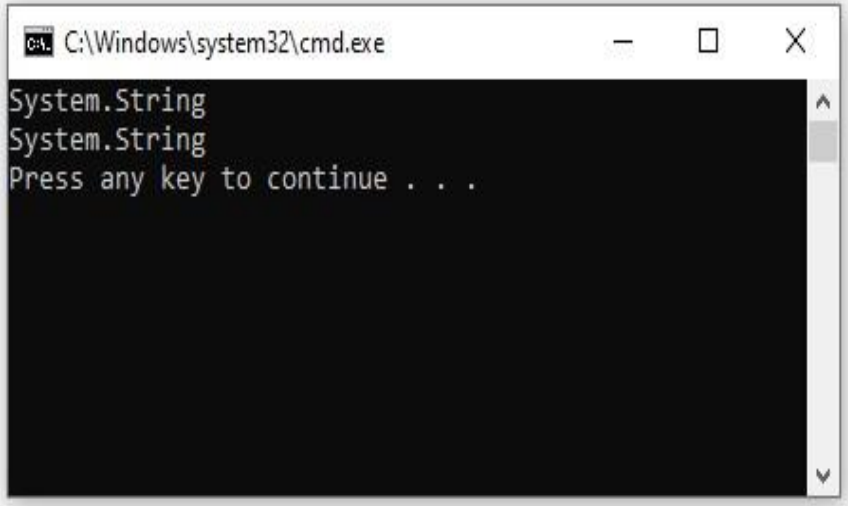
```
using System;

public class Program
{
    public static void Main()
    {
        string str1 = "Hello";
        String str2 = "World!";

        Console.WriteLine(str1.GetType());
        Console.WriteLine(str2.GetType());
    }
}
```

String Vs string C#


```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace String_VS_string
8  {
       0 references
9      class Program
10     {
           0 references
11         static void Main(string[] args)
12         {
13             string str1 = "Hello";
14             String str2 = "World!";
15
16             Console.WriteLine(str1.GetType().FullName); // System.String
17             Console.WriteLine(str2.GetType().FullName); // System.String
18         }
19     }
20 }
21
```



String Vs string C#

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
1   //using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace String_VS_string
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13              String capitalS = "Hello ";
14              string smallS = "World";
15              string st = String.Concat(capitalS, smallS);
16              Console.WriteLine(st);
17          }
18      }
19  }
20
```

String Vs string C#


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

namespace String_VS_string
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            String capitalS = "Hello ";
            string smallS = "World";
            string st = string.Concat(capitalS, smallS);
            Console.WriteLine(st);
        }
    }
}
```

Day 8

1. String Vs string C#
2. C# String Interpolation
3. Lambda Expressions
4. Constructors and Destructors
5. IDisposable Interface

String Interpolation

- ✓ The process of formatting, manipulating, and concatenating the strings is called string interpolation in C# using which expressions and objects can be used as a part of the operation of string interpolation.

String Interpolation

- ✓ This feature of string interpolation was introduced in C# version 6 and before string interpolation was introduced + (plus) operator and String.Format method was used in C# to perform the concatenation operation on strings and by making use of string interpolation, it is possible to place the strings wherever we want them, it is possible to make use of conditions and it is possible to specify the space after or before the string.

String Interpolation

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
string name = “Raghad";
var date = DateTime.Now;

// Composite formatting:
Console.WriteLine("Hello, {0}! Today is {1}, it's
{2:HH:mm} now.", name, date.DayOfWeek, date);

// String interpolation:
Console.WriteLine($"Hello, {name}! Today is
{date.DayOfWeek}, it's {date:HH:mm} now.");
```

Day 8

1. String Vs string C#
2. C# String Interpolation
3. Lambda Expressions
4. Constructors and Destructors
5. IDisposable Interface

Lambda expressions

- ✓ A Lambda Expression in C# is an anonymous function, which contains either an expression or a bunch of statements and the operator used to implement Lambda Expression is '=>'.
- ✓ The Lambda Expression consists of two parts, out of which the left is the input while the right side part is the expression.
- ✓ A Simple Lambda Expression takes in an argument and does return value and one of the most common scenarios to use the lambda expression would the list.

The '=>' is the lambda operator which is used in all lambda expressions. The Lambda expression is divided into two parts, the left side is the input and the right is the expression.

That can be read as "goes to" or "becomes".

The Lambda Expressions can be of two types:

- **Expression Lambda:** Consists of the input and the expression.*Syntax:*

```
input => expression;
```

- **Statement Lambda:** Consists of the input and a set of statements to be executed.*Syntax:*

```
input => { statements };
```

Exercise

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

1- Create List to store 10 numbers.
2- Display the list.
3- Calculate square of each value in the list using lambda expression.
4- Display squares.
5- find all numbers in the list divisible by 3.
6- Display divisible by 3.

Exercise (Solution)


```
static void Main(string[] args)
        {
            // List to store numbers
            List<int> numbers = new List<int>() {36, 71, 12,
                                15, 29, 18, 27, 17, 9, 34};
            // foreach loop to display the list
            Console.Write("The list : ");
            foreach (var value in numbers)
            {
                Console.Write("{0} ", value);
            }
            Console.WriteLine();
            // Using lambda expression to calculate square
           // of each value in the list
            var square = numbers.Select(x => x * x);
            // foreach loop to display squares
            Console.Write("Squares : ");
            foreach (var value in square)
            {
                Console.Write("{0} ", value);
            }
```

Exercise (Solution)

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
            Console.WriteLine();
            // Using Lambda exression to
            // find all numbers in the list
            // divisible by 3
            List<int> divBy3 = numbers.FindAll(x => (x % 3) == 0);

            // foreach loop to display divBy3
            Console.Write("Numbers Divisible by 3 : ");
            foreach (var value in divBy3)
            {
                Console.Write("{0} ", value);
            }
            Console.WriteLine();
        }
    }
}
```

Example 2:

- **Lambda expressions can also be used with User-defined classes.**

Lambda expressions


```
class Student
{

    // properties rollNo and name
    public int rollNo
    {
        get;
        set;
    }

    public string name
    {
        get;
        set;
    }
}
```

Lambda expressions


```
// Main Method
   static void Main(string[] args)
   {
       // List with each element of type Student
       List<Student> details = new List<Student>() {
               new Student{ rollNo = 1, name = "Nivein" },
               new Student{ rollNo = 2, name = "Shoruq" },
               new Student{ rollNo = 3, name = "Lilas" },
               new Student{ rollNo = 4, name = "Aseel" },
               new Student { rollNo = 5, name = "Awadh" }
       };

       // To sort the details list
       // based on name of student
       // in acsending order
       var newDetails = details.OrderBy(x => x.name);

       foreach (var value in newDetails)
       {
           Console.WriteLine(value.rollNo + " " + value.name);
       }
   }
```

Delegate Type:

Any lambda expression can be converted to a **delegate type**. The delegate type to which a lambda expression can be converted is defined by the types of its parameters and return value. If a lambda expression doesn't return a value, it can be converted to one of the Action delegate types; otherwise, it can be converted to one of the Func delegate types.

Action Type:

```
Action<string> greet = name =>
{
        string greeting = $"Hello {name}!";
        Console.WriteLine(greeting);
};

 greet("World");
// Output:
// Hello World!
```

Function Type

```
Func<double, double> cube = x => x * x * x;
```

```
Func<int, int, bool> testForEquality = (x, y) => x = y;
```

Day 8

1. String Vs string C#
2. C# String Interpolation
3. Lambda Expressions
4. **Constructors And Destructors**
5. IDisposable interface

Constructors And Destructors

- Constructors are special methods, used when instantiating a class.
- A constructor can never return anything.
- Several constructors, with the same name, but different parameters.

Constructors And Destructors


```
class Teacher
        {
            string name;
            int age;
            public string Name { get => name; set => name = value;
}
            public int Age { get => age; set => age = value; }

            public Teacher()
               {
                name = "";
                age = 0;
                    Console.WriteLine("I'am Teacher");
            }

            public Teacher(int age)
            {
                this.age = age;
                Console.WriteLine("My Age is: {0}", age);
            }

            public Teacher(string name, int age)
            {
                this.name = name;
                this.age = age;
                Console.WriteLine("My Name is: {0}", name);
            }
        }
```

A constructor can call another constructor.

```
public Teacher()
            {
                Console.WriteLine("I'am Teacher");
                name = "";
                age = 0;
            }

public Teacher(int age) : this()
            {
                this.age = age;
                Console.WriteLine("My Age is: {0}", age);
            }
```

Destructors (~)

- ✓Destructor is method called once an object is disposed.
- ✓Destructor cannot be inherited.
- ✓Destructor does not have parameters.
- ✓Execution of the destructor for the instance may occur at any time after the instance becomes eligible for destruction.

Destructors (~)

```
~Teacher()
{
    Console.WriteLine("Destructor was called");
}
```

Constructors and Destructors Example

Constructors And Destructors

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
class Person
   {
       public Person()
       { Console.WriteLine("Person say: Hi");}

       ~Person()
       { Console.WriteLine("Person destructor was called");}
   }
   class Teacher : Person
   {
       string name;
       int age;
       public string Name { get => name; set => name = value; }
       public int Age { get => age; set => age = value; }
       public Teacher()
       {
           Console.WriteLine("Teacher say: Hi");
           name = "";
           age = 0;
       }
       public Teacher(int age) : this()
       {
           this.age = age;
           Console.WriteLine("My Age is: {0}", age);
       }
       public Teacher(string name, int age) : this(age)
       {
           this.name = name;
           Console.WriteLine("My Name is: {0}", name);
       }
       ~Teacher()
       { Console.WriteLine("Teacher destructor was called");}
   }
```

Destructor Used to:

- ✓Cleanup resources used by the object.
- ✓Memory management.
- ✓It invokes the finalize method which removes all the instance of objects that is no longer in use.

Garbage collected

- ✓The framework will free the objects that you no longer use.
- ✓There may be times where you need to do some manual cleanup.

Managed & Unmanaged resources

✓Managed resources basically means "managed memory" that is managed by the garbage collector.

✓Unmanaged resources are then everything that the garbage collector does not know about. For example:

- Open files
- Open network connections
- Sockets

Day 8

1. String Vs string C#
2. C# String Interpolation
3. Lambda Expressions
4. Constructors and Destructors
5. **IDisposable Interface**

IDisposable Interface

IDisposable Interface

- Provides a mechanism for releasing unmanaged resources.

```
public interface IDisposable
{
    void Dispose();
}
```

IDisposable Interface (Example)


```
 using (StreamWriter file = new
StreamWriter(@"C:\Users\Lilas\Desktop\Try\file7.txt"))
{
    file.WriteLine("lilas");
    file.Dispose();
}
```

IDisposable interface Example

IDisposable Interface (Example)

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
class Library : IDisposable
    {
        string name;

        StreamReader file = new
StreamReader(@"C:\Users\Lilas\Desktop\Try\file.txt");

        public string Name { get => name; set => name = value; }
        public Library()
        {
            name = "";
            Console.WriteLine("Library is open");
        }
        public void Dispose()
        {
            Console.WriteLine("Library Close");
            Dispose(true);
            GC.SuppressFinalize(this);
        }
```

IDisposable Interface


```
        private bool disposed = false;
        protected virtual void Dispose(bool disposing)
        {
            if (!this.disposed)
            {
                if (disposing)
                {
                    // Dispose managed resources.
                    file.Dispose();
                }
                // Note disposing has been done.
                disposed = true;
                FreeChunk();
                file = null;
            }
        }

        bool isFreed = false;
        private void FreeChunk()
        {
            if (isFreed)
                return;
            //Unmanaged memory
            file.Close();
        }

        ~Library()
        {
            Dispose(true);
            Console.WriteLine("Library Close in Destructor");
        }
    }
```

Exercise

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Create the ZooAnimal Class with two constructor functions and destructor.

The first function is the default and the secound function has three parameters: string followed by two integer parameters.

Copy the string parameter into the name field, and then assign the two integer parameters to cageNumber and numAnimals respectively.

Create the Cat class inherited from ZooAnimal with constructor and destructor.

Day Eight Task

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

On the E-Learning Portal