

## Sec 15 : System.Object Class

14 July 2024 16:29



ct class



Harsha Vardhan

- › UI Expert
- › .NET Expert
- › Module Lead
- › Corporate Instructor

 Harsha  
Web University

## System.Object class



**Understanding 'System.Object' class**

What is 'Object' class and its purpose.



**Methods of 'System.Object' class**

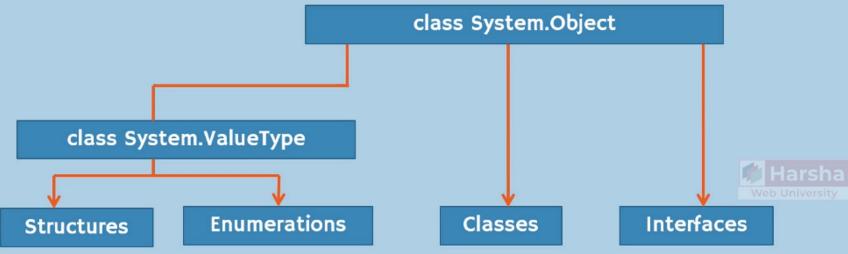
GetType(), ToString() etc.

Introducing System.Object class

Next: Methods of 'Object' class

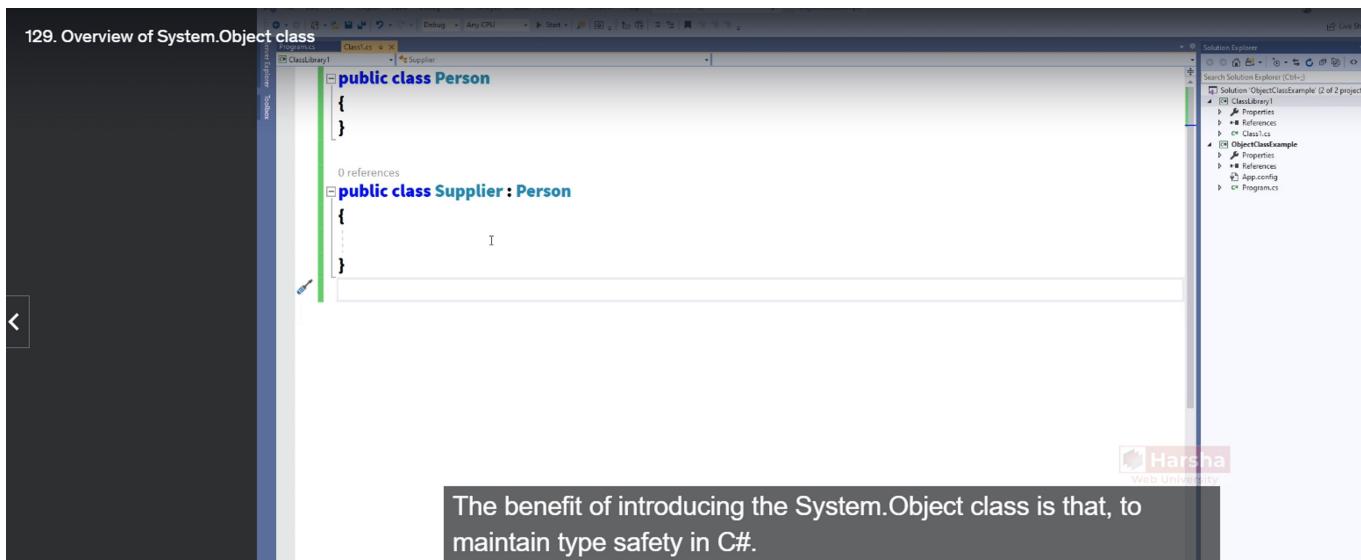
**What**

- › The "System.Object" is a pre-defined class, which is the "Ultimate super class (base class)" in .net.
- › All the classes and other types are inherited from System.Object directly / indirectly.



```
graph TD; SystemObject["class System.Object"] --> ValueType["class System.ValueType"]; SystemObject --> Classes["Classes"]; SystemObject --> Interfaces["Interfaces"]; ValueType --> Structures["Structures"]; ValueType --> Enumerations["Enumerations"];
```

 Harsha  
Web University



That means in a variable of particular type you must be able to assign only the values of the particular type. You should not be able to assign the values other than the particular type. To maintain this kind of type safety, the System.Object class is introduced in C#.

So all over in the entire language there is no any type which is not child of System.Object class. Because it must be child of System.Object class either directly or indirectly.

```
class Program
{
    static void Main()
    {
        Person p;
        System.Object abc = new Person();
        p = new Supplier();

        int a = 10;
    }
}
```

## Methods of 'System.Object' class

### System.Object Class [Pre-defined]

```
namespace System
{
    class Object
    {
        virtual bool Equals( object value );
        virtual int GetHashCode();
        Type GetType();
        virtual string ToString();
    }
}
```



while working with collections such as

## Methods of Object class

### Methods of System.Object class

#### bool Equals( object value )

- Compares the current object with the given argument object; returns true, if both are same objects; returns false, if both are different objects.

#### int GetHashCode( object value )

- Returns the a number that represents the object. It is not guarantee that the hash code is unique, by default.

#### Type GetType()

- Returns the name of the class (including namespace path), based on which, the object is created.

#### string ToString()

- By default, it returns the name of the class (including namespace path), based on which, the object is created.

- It is virtual method, which can be overridden in the child class.

## Understanding and Overriding Methods of Object Class

The screenshot shows a Microsoft Visual Studio code editor window. The title bar says "Class Library - C:\C#\Deep\ObjectClassExample\ObjectClassExample\Program.cs". The code in the editor is:

```
public class Person
{
    //properties
    public string PersonName { get; set; }
    public string Email { get; set; }
}
```

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `Program.cs` with the following content:

```

using System;

class Program
{
    static void Main()
    {
        //create an object of Person class
        System.Object obj = new Person() { PersonName = "Scott", Email = "scott@gmail.com" };

        //access methods
        Console.WriteLine(obj.Equals(new Person() { PersonName = "Scott", Email = "scott@gmail.com" }));
        Console.ReadKey();
    }
}

```

The output window on the right shows the result of the execution: `False`.

We're getting false as they are 2 separate object even though they have same value.  
but we can change it.

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `Person.cs` with the following content:

```

public class Person
{
    //properties
    public string PersonName { get; set; }
    public string Email { get; set; }

    //overriding Equals method
    public override bool Equals(object obj)
    {
        Person p = (Person)obj;
        if(this.PersonName == p.PersonName && this.Email == p.Email)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

A red arrow points to the `Equals` method definition.

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `Program.cs` with the following content:

```

using System;

class Program
{
    static void Main()
    {
        //create an object of Person class
        System.Object obj = new Person() { PersonName = "Scott", Email = "scott@gmail.com" };

        //access methods
        Console.WriteLine(obj.Equals(new Person() { PersonName = "Scott", Email = "scott@gmail.com" }));
        Console.ReadKey();
    }
}

```

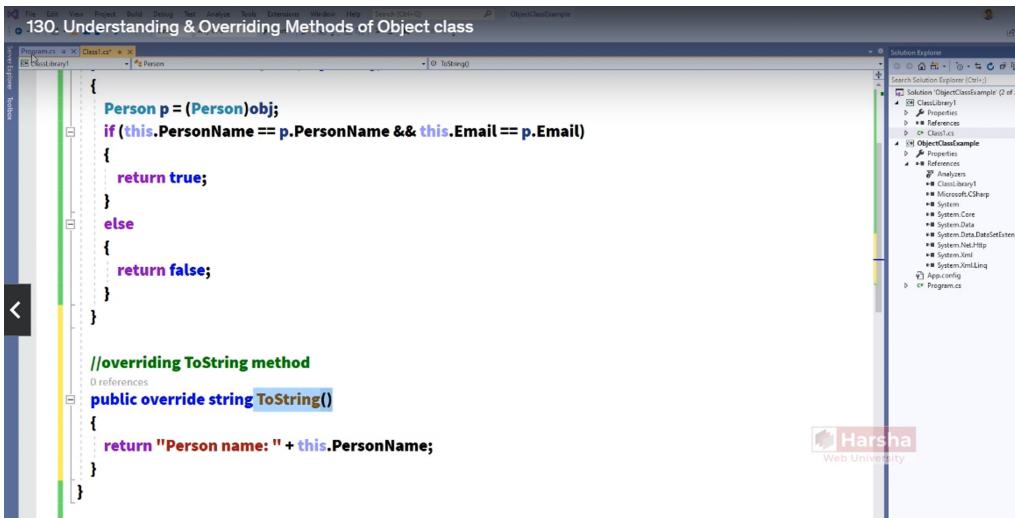
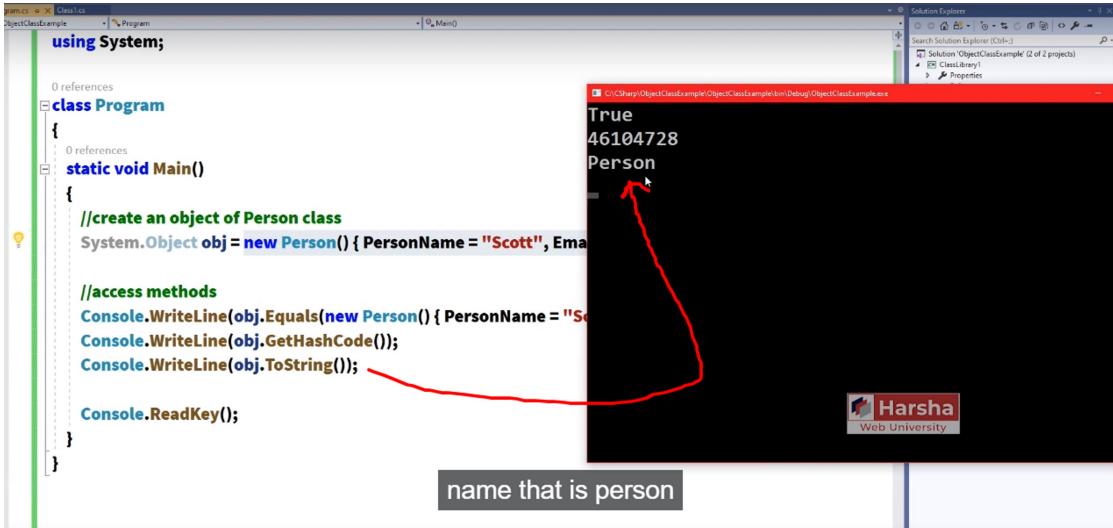
The output window on the right shows the result of the execution: `True`.

## 130. Understanding & Overriding Methods of Object class

### Methods of System.Object class

- > **bool Equals( object value )**
  - > Compares the current object with the given argument object; returns true, if both are same objects; returns false, if both are different objects.
- > **int GetHashCode( object value )**
  - > Returns the a number that represents the object. It is not guarantee that the hash code is unique, by default.
- > **Type GetType( )**
  - > Returns the name of the class (including namespace path), based on which, the object is created.
- > **string ToString( )**
  - > By default, it returns the name of the class (including namespace path), based on which, the object is created.
  - > It is virtual method, which can be overridden in the child class.

1x 9:31 / 13:35



The screenshot shows the Microsoft Visual Studio IDE. On the left is the code editor with the following C# code:

```
using System;

class Program
{
    static void Main()
    {
        //create an object of Person class
        System.Object obj = new Person() { PersonName = "Scott", Email = "scott@dotnetperls.com" };

        //access methods
        Console.WriteLine(obj.Equals(new Person() { PersonName = "Scott" }));
        Console.WriteLine(obj.GetHashCode());
        Console.WriteLine(obj.ToString());
        Console.WriteLine(obj.GetType().ToString());

        Console.ReadKey();
    }
}
```

The output window on the right displays the results of the console application:

```
True
46104728
Person name: Scott
Person
to string
```

`ToString()` can be overridden but `GetType().ToString()` always return class name and can not be overridden.

**About Object class**

**Lightbulb icon:**

- › All C# classes, structures, interfaces, enumerations are children of `System.Object` class.
- › Every method defined in the `Object` class is available in all objects in the system as all classes in the .NET Framework are derived from `Object` class.
- › Derived classes can override `Equals`, `GetHashCode` and `ToString` methods of `Object` class.
- › `System.Object` class is meant for achieving "type safety" in C#.

**Harsha Web University**



Harsha Vardhan

- › UI Expert
- › .NET Expert
- › Module Lead
- › Corporate Instructor



## Boxing & Unboxing

### Agenda

1

#### Boxing

Conversion from Value-type to Reference-type.

2

#### Unboxing

Conversion from Reference-type to Value-type.

### Boxing

### Next: Unboxing

#### What

- › It is a process of converting a value from "Value-Type Data Type" to "Reference-Type Data Type", if they are compatible data types.

#### How

- › This can be done automatically [no need of any syntax].



```
Program.cs
class Program
{
    static void Main()
    {
        //primitive variable
        int x = 10;

        //boxing (value-type to reference-type)
        object obj = x;

        System.Console.WriteLine(x); //Output: 10
        System.Console.WriteLine(obj); //Output: 10
        System.Console.ReadKey();
    }
}
```

10  
10

But overall, in the output there is no any kind of difference.

So Boxing is very common process in C# wherever you are trying to assign the value of primitive value into an object of System.Object class, it is called Boxing.

Generally in the real programming, you don't care about Boxing; but it happens internally.

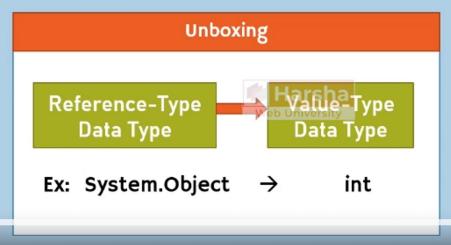
## Unboxing

### What

- It is a process of converting a value from "Reference-Type Data Type" to "Value-Type Data Type", if they are compatible data types.

### How

- This should be done explicitly (by using explicit casting).
- Syntax: (DestinationDataType)SourceValue
- Example: (short)100



## Unboxing

**What**

- It is a process of converting a value from "Reference-Type Data Type" to "Value-Type Data Type", if they are compatible data types.

**How**

- This should be done explicitly (by using explicit casting).
- Syntax:** (DestinationDataType)SourceValue
- Example:** (short)100

Stack

reference variable

address

primitive variable

value

Heap

object

COPY

value

Unboxing

Reference-Type Data Type

Value-Type Data Type

Ex: System.Object → int

If the value has been copied from the heap into the stack, then it is called as Unboxing.

```

class Program
{
    static void Main()
    {
        //reference type variable
        object obj = 10;

        //unboxing (reference-type to value-type)
        int x = (int)obj;

        System.Console.WriteLine(x); //Output: 10
        System.Console.WriteLine(obj); //Output: 10
        System.Console.ReadKey();
    }
}

```