

**C#-8**

# Abstraction

## Abstraction

- Abstraction is a process of defining the essential concepts while ignoring the non-essential details.
- It is the process of picking out (*abstracting*) common features of objects and procedures.
- Abstraction is selective ignorance
  - Decide what is important and what is not
  - Focus and depend on what is important
  - Ignore and do not depend on what is unimportant
  - Use encapsulation to enforce an abstraction

The purpose of abstraction is not to be vague,  
but to create a new semantic level in which one can be absolutely  
precise.  
- Edsger Dijkstra

Its a Internal Implementation which you are trying to show to the external person  
SO the Information which you are trying to expose>Show  
Hiding the implementation Showing the Feature of a particular object

So Abstraction is a process where you **show**, ignore the implementation or hide the non essential things and show the important things

SO it is showing the important things not the hiding things

**You cannot say that i'm hiding my body with the shirt it dosen't make sense, i'm wearing shirt that's it**

Which i can see is Abstraction, Which i cannot see is its Implementation

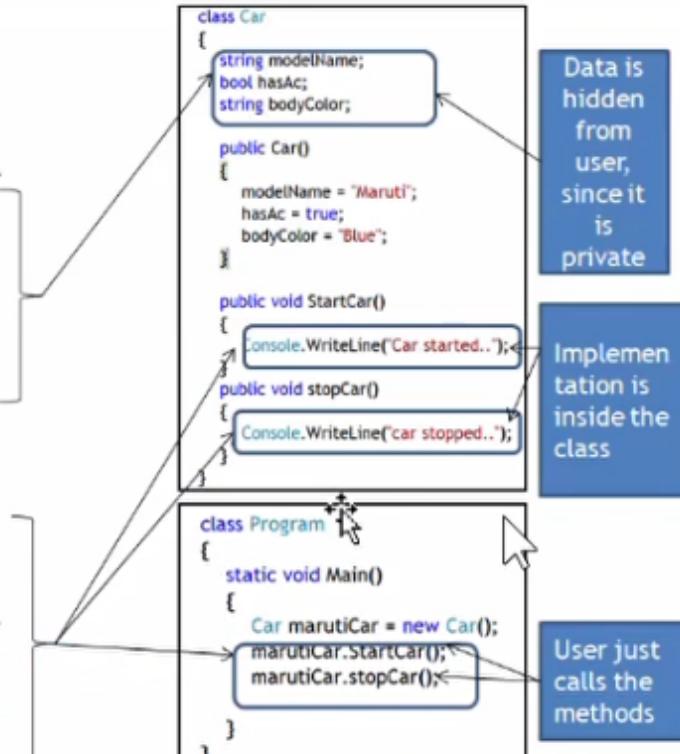
## Different Types of Abstraction

- Data Abstraction

- Programming languages define constructs to simplify the way information is presented to the programmer.
- Fields are declared with proper access specifiers, so that they are hidden from users. So, users can't access them directly

- Functional Abstraction

- Programming languages have constructs that 'gift wrap' very complex and low level instructions into instructions that are much more readable. The method implementation details are hidden from user

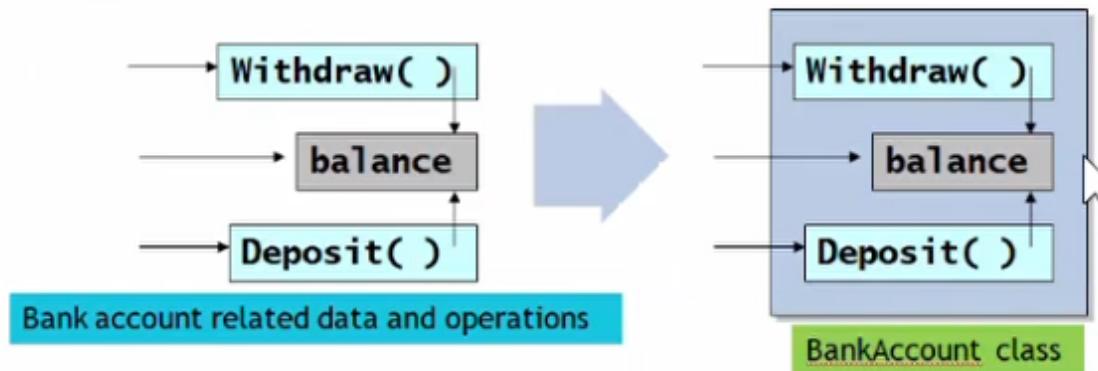


If We want our data members to be hidden this will be my data abstraction  
there can be some of the methods which i want to show so the data will be hidden only the method will be exposed

## What is Encapsulation?

### 1. Combining Data and Methods

- Combine the data and methods in a single *capsule*
- The capsule boundary forms an inside and an outside



Consider a bank account, where account maintains a balance and also supports behavior such as, withdrawal and deposit of money.

there will be lot of abstractions which we can write for example withdraw, deposit i can expose but Balance i can always Encapsulate withdraw, deposit i can expose to the user but whereas balance i can hide it So there will be calculate method and the withdraw and deposit method will use this calculate method

The diagram illustrates encapsulation through two representations of a `BankAccount` class. On the left, a blue rounded rectangle contains a green `Withdraw( )` box, a brown `balance` box, and a green `Deposit( )` box, all connected by arrows. A pink arrow points to the right, leading to another blue rounded rectangle. Inside this second rectangle, the same three boxes are shown, but the brown `balance` box now has a red 'X' over it and is not connected to the other boxes, while the `Withdraw( )` and `Deposit( )` boxes remain connected.

What is Encapsulation?

2. Controlling Access Visibility

- Methods are *public*, accessible from the outside
- Data is *private*, accessible only from the inside

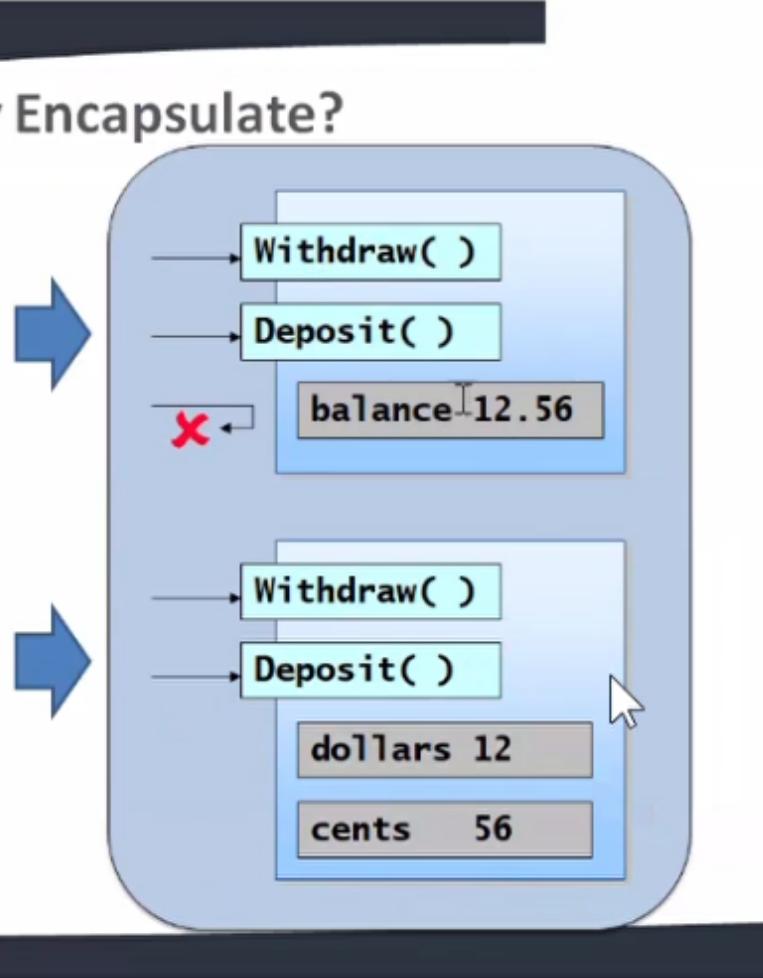
BankAccount class with just data and operations      BankAccount class with private data and public operations

If you declare `balance` as private field then you are not allowing `balance` to be used directly from outside the class, thus essentially blocking the view of the variable.

So remember you should not have this `balance` as public that will always be a private access

## Why Encapsulate?

- Allows control
  - Use of the object is solely through the public methods
- Allows change
  - Use of the object is unaffected if the private data type changes



What will happen if i expose this balance to the external world so anybody will go and modify it and that lead to soo much of problems

# Properties

- Another mechanism to encapsulate fields
- Properties provide:
  - A useful way to encapsulate information inside a class
  - Concise syntax
  - Flexibility
- Properties provide field-like access
  - Use get accessor statements to provide read access
  - Use set accessor statements to provide write access

```
class Car
{
    string modelName;
    public string ModelName
    {
        get { return modelName; }
        set { modelName = value; }
    }

    bool hasAc;
    public bool HasAc
    {
        get { return hasAc; }
        set { hasAc = value; }
    }

    string bodyColor;
    public string BodyColor
    {
        get { return bodyColor; }
        set { bodyColor = value; }
    }
}
```

We have already discussed property tags  
There is **no memory allocation for a property**

## How to use Property?

- Set value of private fields through property (actually calling set accessor)

- Return value from private fields through property (actually get accessor)

```
class Program
{
    static void Main()
    {
        Car marutiCar = new Car();
        //assigning value to fields using properties
        //calling set accessor
        marutiCar.BodyColor = "Blue";
        marutiCar.HasAc = true;
        marutiCar.ModelName = "Maruti 800";
        //calling getstring Car.ModelName
        Console.WriteLine("Which Model? " + marutiCar.ModelName);
        Console.WriteLine("Conatains A/C? " + marutiCar.ModelName);
        Console.WriteLine("What Color? " + marutiCar.ModelName);
    }
}
```

## Property Types

- Read/write properties
  - Have both `get` and `set` accessors
  - Sets value to and gets value from a field
- Write-only properties – very limited use
  - Have `set` accessor only
  - Just sets the value to a field
- Read-only properties
  - Have `get` accessor only
  - Just returns value of some field

```
class Employee
{
    private decimal basicPay;
    public decimal BasicPay
    {
        set { basicPay = value; }
    }
    private decimal hraPay;
    public decimal HraPay
    {
        set { hraPay = value; }
    }
    private decimal daPay;
    public decimal DAPay
    {
        set { daPay = value; }
    }
    private decimal totalSalary;
    public decimal TotalSalary
    {
        get { return totalSalary; }
    }
    public void CalculateSalary()
    {
        totalSalary = basicPay + daPay + hraPay;
    }
}
```

## Comparing Properties to Fields and Methods

- Properties are “logical fields”
  - The `get` accessor can return a computed value
- Similarities
  - Syntax for creation and use is the same
- Differences
  - Properties are not values; they have no address
  - Properties cannot be used as `ref` or `out` parameters to methods

- Similarities
  - Both contain code to be executed
  - Both can be used to hide implementation details
  - Both can be virtual, abstract, or override
- Differences
  - Syntactic – properties do not use parentheses
  - Semantic – properties ~~can~~ not be `void` or take arbitrary parameters

## Class and Class Member Access Modifiers

Access Modifier	Description	Applied On
abstract	If used with any member of the class, then it means that the member is without an implementation in the base class and will be provided with an implementation in the derived class	Class and class members
virtual	Member of the class declared with this keyword may or may not be given a new implementation in the derived class	Only class members
override	This keyword is used to mention that a new implementation is being provided to the virtual or an abstract member inherited from base class	Only class members
sealed	If used with a class then the class can't be derived and if used with a member of the class the member can't be inherited	Class and class members

## Class and Class Member Access Modifiers

Modifier	Description	Applied On
static	If used with a class, then the class contains all static members and can't contain any non-static members and also an instance of the class can't be created. If declared with a member then the member can't be accessed using instance of the class, rather can be accessed using the class name. static members are 'shared' amongst different instances of the class	Class and class members
<u>readonly</u>	Assignments to the fields can only occur in the declaration or in the same class constructor	Only on fields of a class 
Const	Assignments to the fields can only occur in the declaration	Only on fields of a class

## Class and Class Member Access Specifiers

Modifier	Description	Applied On
volatile	Field can be modified in the program by something like the operating system, the hardware, or a concurrently executing thread	Only on fields of a class
extern	Indicates that the method is implemented externally	Class members
new	Allows you to declare a member with the same name or signature as an inherited member.	Class members

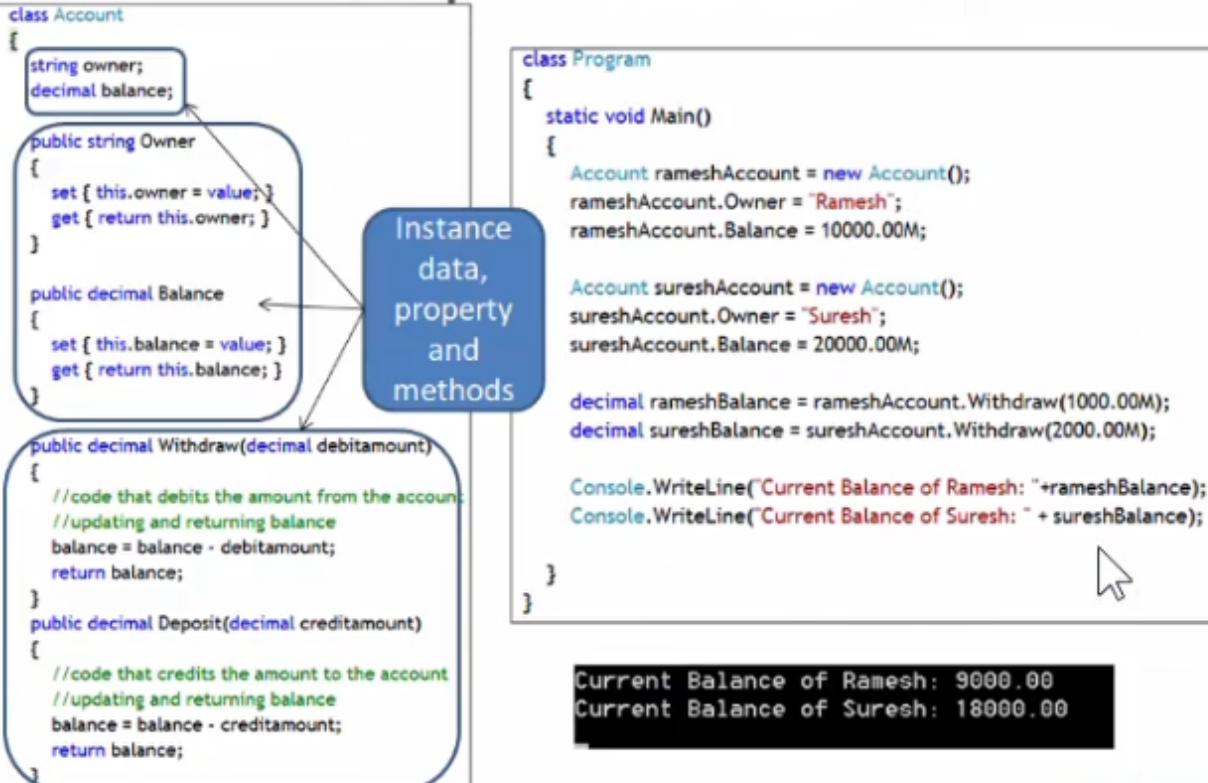


## Object/Instance Data

- Object data describes information for *individual* objects
  - For example, each bank account has its own balance. If two accounts have the same balance, it is only a coincidence.
  - Instance data is accessed through the instance of the class



## Example: Instance Member



## Using this Keyword

- ‘this’ is a reference to the object of the current class
  - It can be used to distinguish instance variables from local variables
  - It can be assigned to other references, or passed as a parameter, or cast to other types
  - It cannot be used in a static contextUseful when identifiers from different scopes clash

The diagram illustrates the use of the 'this' keyword through two code snippets. The top snippet shows a class definition for 'Car' with three properties: 'modelName', 'hasAc', and 'bodyColor'. Each property has a get and set accessor. The bottom snippet shows a constructor for 'Car' that initializes these properties using the 'this' keyword to distinguish them from parameters with the same names.

```
class Car
{
    string modelName;
    public string ModelName
    {
        get { return this.modelName; }
        set { this.modelName = value; }
    }

    bool hasAc;
    public bool HasAc
    {
        get { return this.hasAc; }
        set { this.hasAc = value; }
    }

    string bodyColor;
    public string BodyColor
    {
        get { return this.bodyColor; }
        set { this.bodyColor = value; }
    }
}

class Car
{
    string modelName;
    bool hasAc;
    string bodyColor;

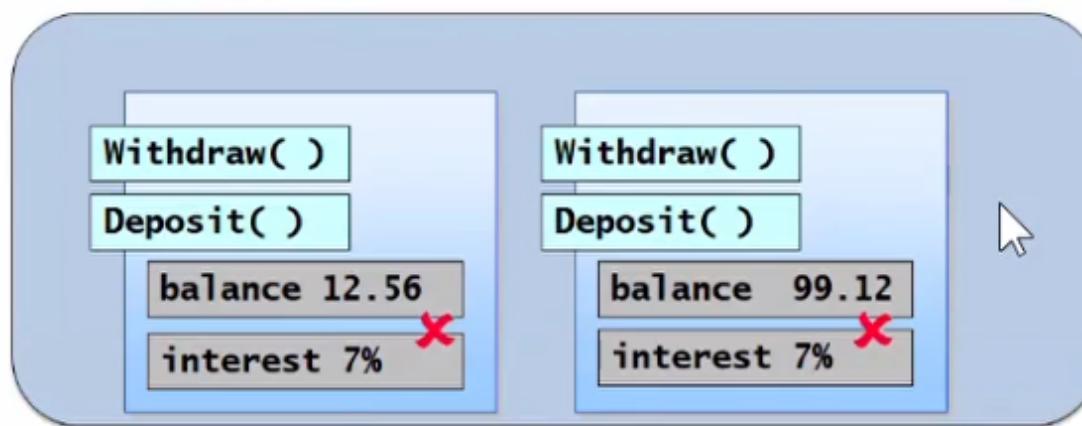
    public Car(string name, bool ac, string color)
    {
        this.modelName = name;
        this.bodyColor = color;
        this.hasAc = ac;
    }
}
```

If this statement were  
modelname = modelname;  
What would happen?

## Static Members

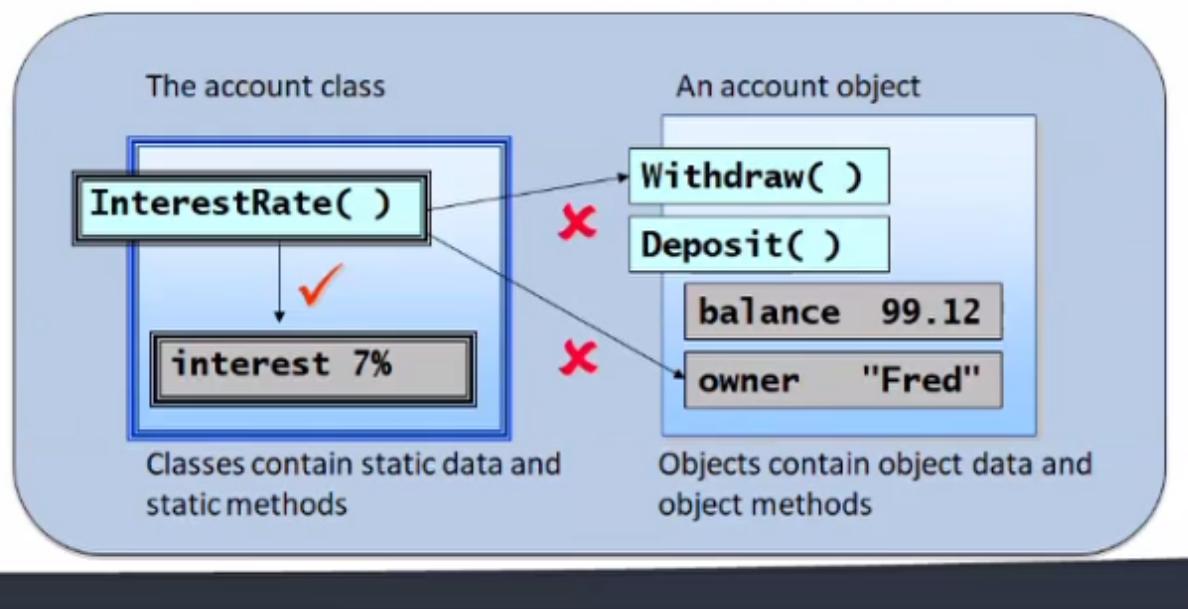
## Using Static Data

- Static data describes information for *all* objects of a class
  - For example, suppose all accounts share the same interest rate. Storing the interest rate in every account would be a bad idea.
  - Make it a static data

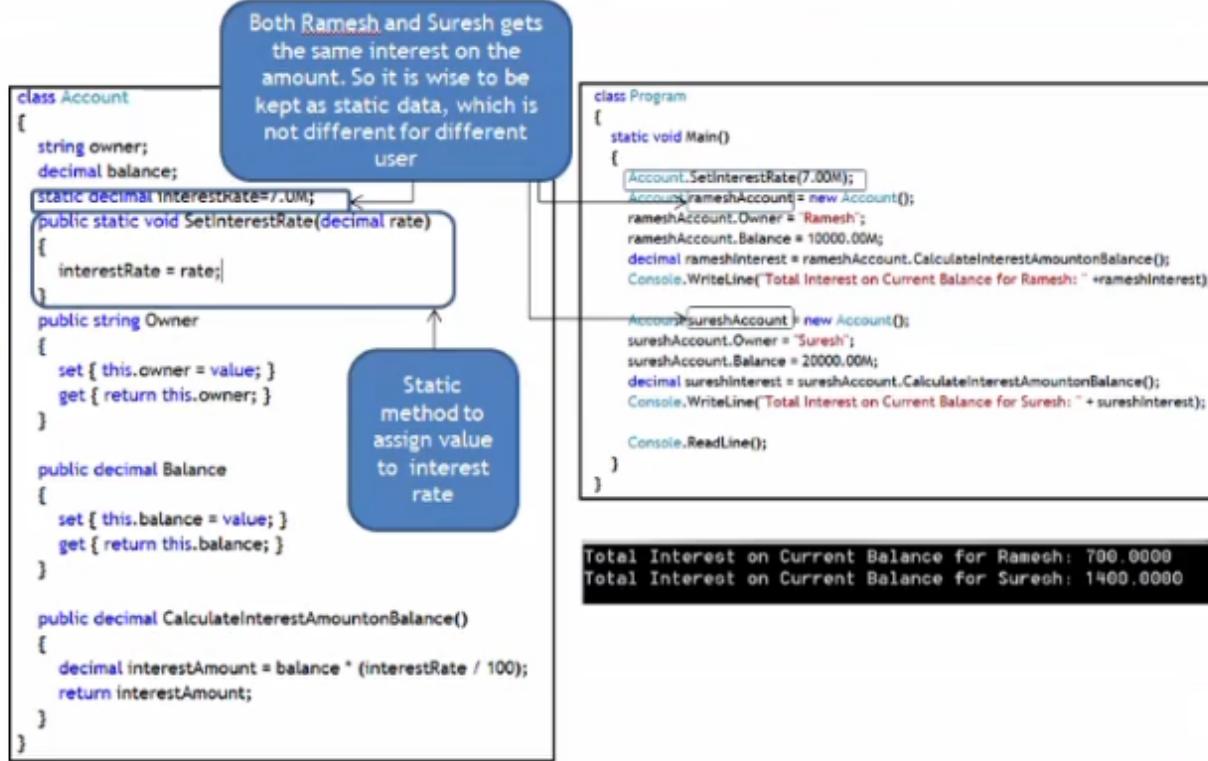


## Using Static Methods

- Static methods can only access static data
  - A static method is called on the class, not the object



## Example: Static Data Member



## Static Constructor

- Constructor can be static, too.
- Purpose
  - Called by the class loader at run time
  - Called only once in the application lifetime, no matter how many objects are created. It is called just before the first object creation.
  - Guaranteed to be called before instance constructor
  - At least one object of the class has to be created, so that at least once the instance constructor gets called
  - Can be used to initialize static fields
- Restrictions
  - Cannot be called
  - Cannot have an access modifier
  - Must be parameter less (i.e., can't be overloaded)

## Example:

```
class Account
{
    string owner;
    decimal balance;
    static decimal interestRate;

    static Account()
    {
        interestRate = 7.0M;
        Console.WriteLine("Interest rate is: " + interestRate);
    }

    public string Owner
    {
        get { return owner; }
        set { owner = value; }
    }

    public decimal Balance
    {
        get { return balance; }
        set { balance = value; }
    }

    public decimal CalculateInterestAmountonBalance()
    {
        decimal interestamount = balance * (interestRate / 100);
        return interestamount;
    }
}
```

Instead of a static method, 'interestRate' variable is being assigned in a static constructor

```
class Program
{
    static void Main(string[] args)
    {
        Account rameshAccount = new Account();
        rameshAccount.Owner = "Ramesh";
        rameshAccount.Balance = 10000.00M;
        decimal rameshInterest = rameshAccount.CalculateInterestAmountonBalance();
        Console.WriteLine("Total Interest on Current Balance for Ramesh: " + rameshInterest);

        Account sureshAccount = new Account();
        sureshAccount.Owner = "Suresh";
        sureshAccount.Balance = 20000.00M;
        decimal sureshInterest = sureshAccount.CalculateInterestAmountonBalance();
        Console.WriteLine("Total Interest on Current Balance for Suresh: " + sureshInterest);

        Console.ReadLine();
    }
}
```

```
Interest rate is: 7.0
Total Interest on Current Balance for Ramesh: 700.0000
Total Interest on Current Balance for Suresh: 1400.0000
```

Notice, static constructor has been executed only once, though two objects of Account class has been created

## Comparison Between Static and Non-static data

Static Member	Non-Static Member
1. Static data and static method are not accessed through instances, rather through class name	1. Non-static data and method are accessed through instances, not using class name
2. Only single copy of static data member will be present and every instance will share that static data	2. Non-static data is different for different instances and it is not shared amongst different instances. It is particular to the instance
3. Static method can only access static data	3. Non-static method can access static as well as non-static data

What are static members

StaticDemo.cs\* ✘

ConsoleApp

ConsoleApp.StaticDemo

Show()

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class StaticDemo
10     {
11         int x = 0;
12         static int y = 0;
13
14         void Show()
15         {
16             int z = 0;
17             x++;
18             y++;
19             z++;
20             Console.WriteLine("X = {0} \t Y = {1} \t Z = {2}", x, y, z);
21         }
22     }
}
```

0 references

0 references

121 %

No issues found

Ln: 20 Ch: 60 Col: 69 TABS CRLF

Solution Explorer

Search Solution Explorer (Ctrl+Shift+F)

Solution 'ConsoleApp' (2 of 2 projects)

AccessTest

ConsoleApp

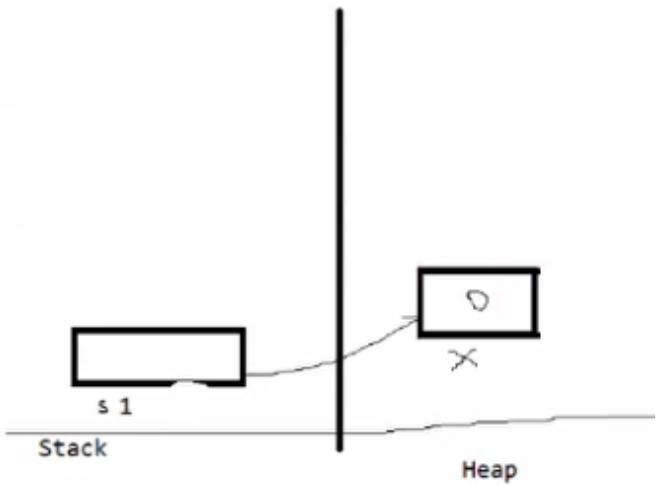
- Properties
- References
- CollectionsDemo
- AccessDemo.cs
- App.config
- ArrayDemo1.cs
- ArrayDemo2.cs
- BreakContinueDemo.cs
- Calc.cs
- ClassDemo.cs
- CommandLineDemo.cs
- ConvertType.cs
- EnumDemo.cs
- GoToDemo.cs
- IfDemo1.cs
- IfDemo2.cs
- IfDemo3.cs
- IncDemo.cs
- InOutDemo.cs
- InOutDemo1.cs
- IntToWord.cs

The screenshot shows a code editor window for a C# file named `StaticDemo.cs`. The code defines a class `StaticDemo` with a `Show()` method and a `Main()` method.

```
13
14     void Show()
15     {
16         int z = 0;
17         x++;
18         y++;
19         z++;
20         Console.WriteLine("X = {0} \t Y = {1} \t Z = {2}", x, y, z);
21     }
22
23     static void Main()
24     {
25         StaticDemo s1 = new StaticDemo();
26         StaticDemo s2 = new StaticDemo();
27
28         s1.Show();
29         s2.Show();
30         s1.Show(); // Cursor is here
31         s2.Show();
32     }
33 }
34 }
```

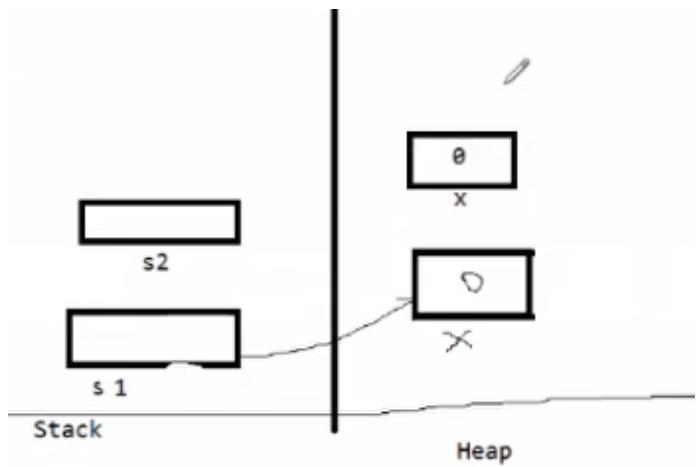
What will be the output of this program ?

initially S1 will only have x nothing else and x is initialize to 0

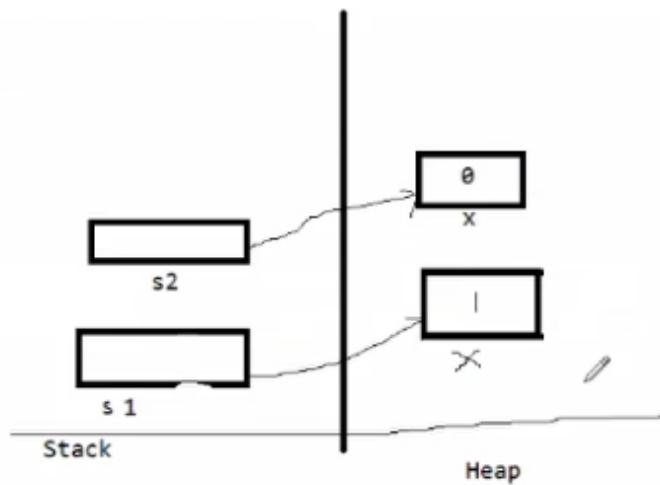


this is what happen in line no 25

int the line no 26 i'll again have one more instance of S2 which will be created and this S2 will have one more variable here and this will have its own x and this will have the value 0 in it and the base address of his will be given



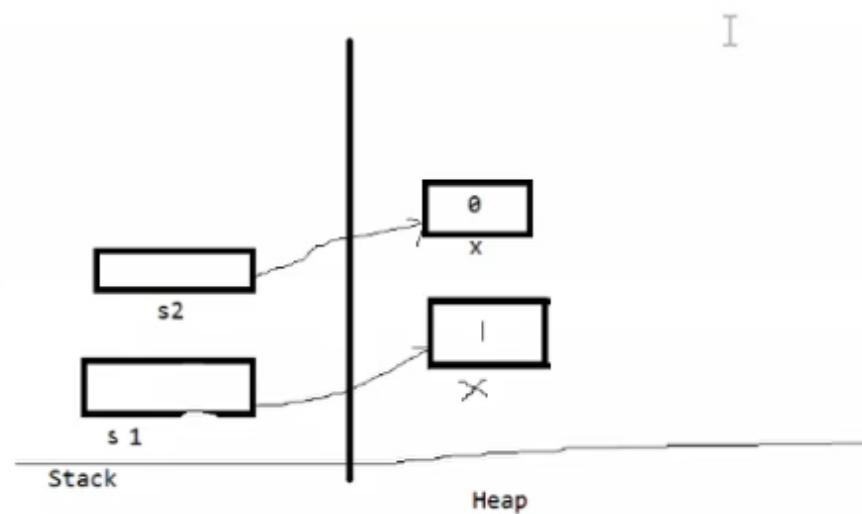
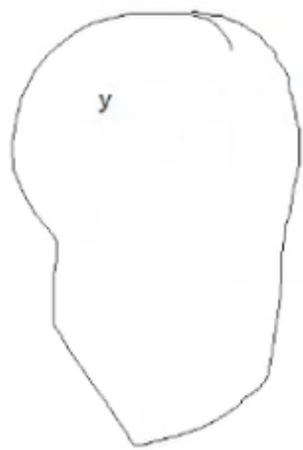
Now when you say `x1.display`, so it will say `x++` so `x` value of this becomes 1



Where is `y` then ?

**Remember y is a static variable so it will be present in a static block**

So we'll have a static block so static block will have a variable y and that will be accessed by all the members of the house



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

namespace ConsoleApp
{
    class StaticDemo
    {
        int x = 0; // Instance Variable
        static int y = 0; // Class Variable
        void Show()
        {
            int z = 0; // Local Variable
            x++;
            y++;
            z++;
            Console.WriteLine("X = {0} \t Y = {1} \t Z = {2}", x, y, z);
        }
    }
}
```

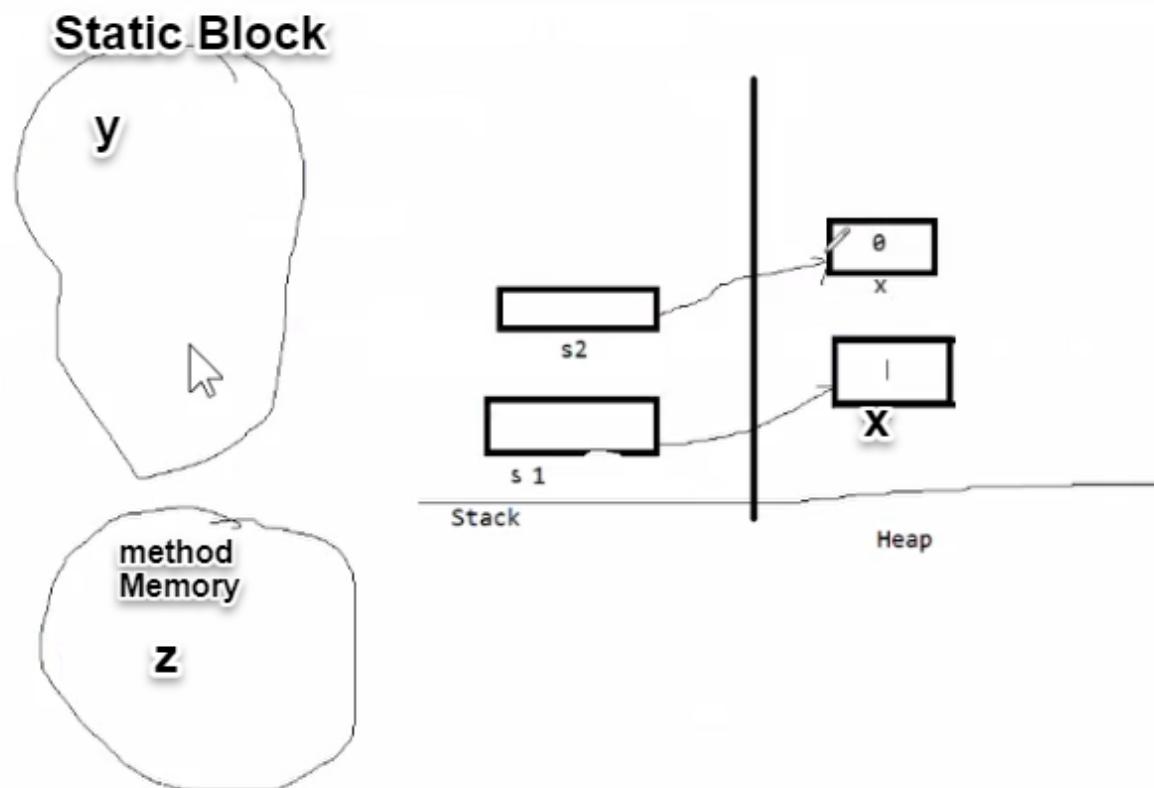
Meaning Every object will have its own Copy

I have my own stomach  
you have your own stomach  
If I'll eat it will not fill your stomach

Z will be present with the method, you'll have method memory

All the method we create here, where all these method will be present, will it be part of object what we create here ----NO

it cannot be part of object imagine i would have written 1000 lines of code and i have such 10 methods so  $10 \times 1000$  becomes 10000 lines of code if all those 10000 lines of code is present inside this object and if i create 2 or 3 object your memory will be filled



so you have a method memory and whenever this object is passed when you call [s1.show\(\)](#) internally it will pass **this** and this will be tagged with that particular object, so your operation system is smart enough it knows which method has been called or CLR will know which method is been called

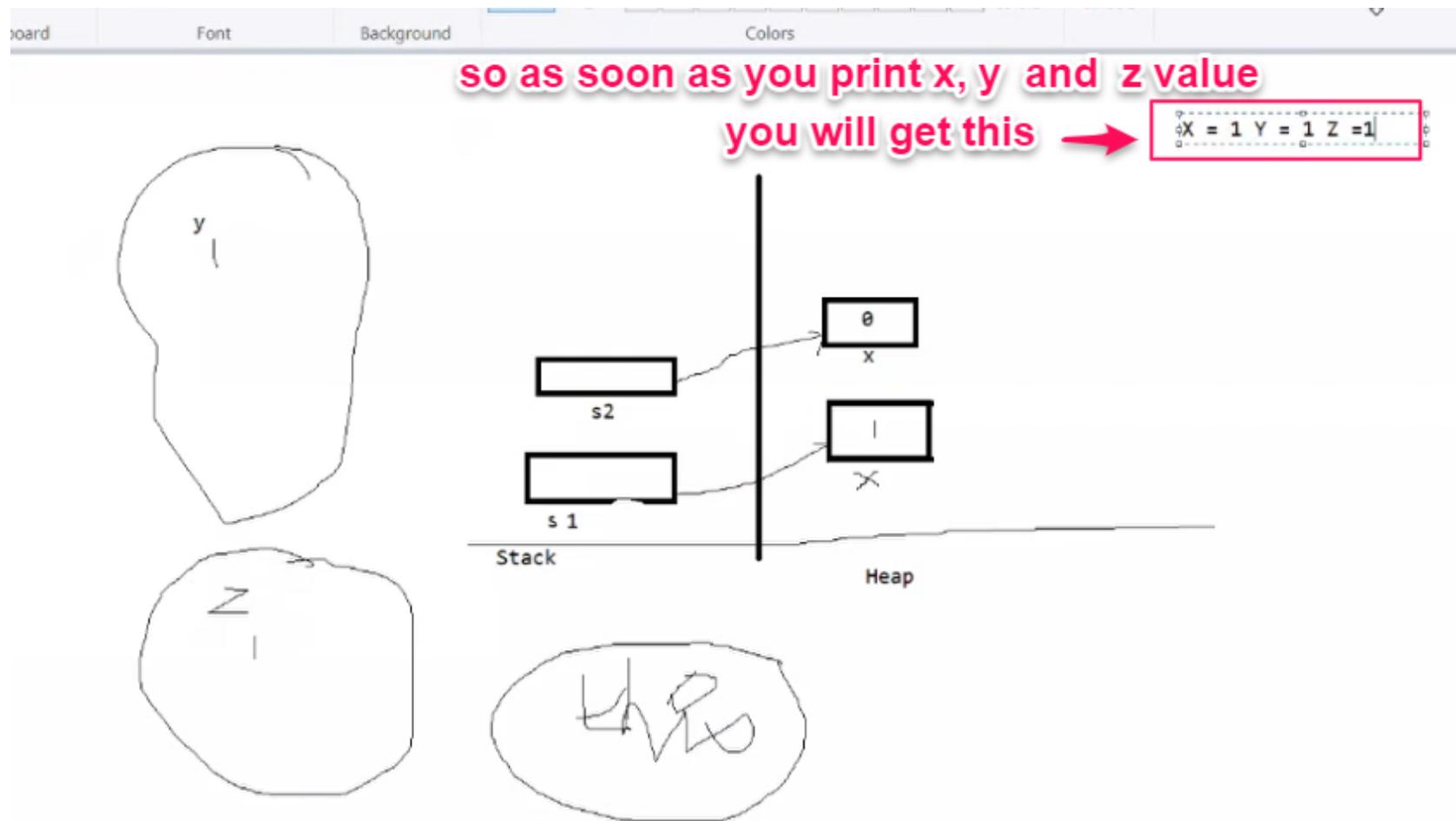
So this will act as a pointer

So if you call s1 so this will be pointing to s1

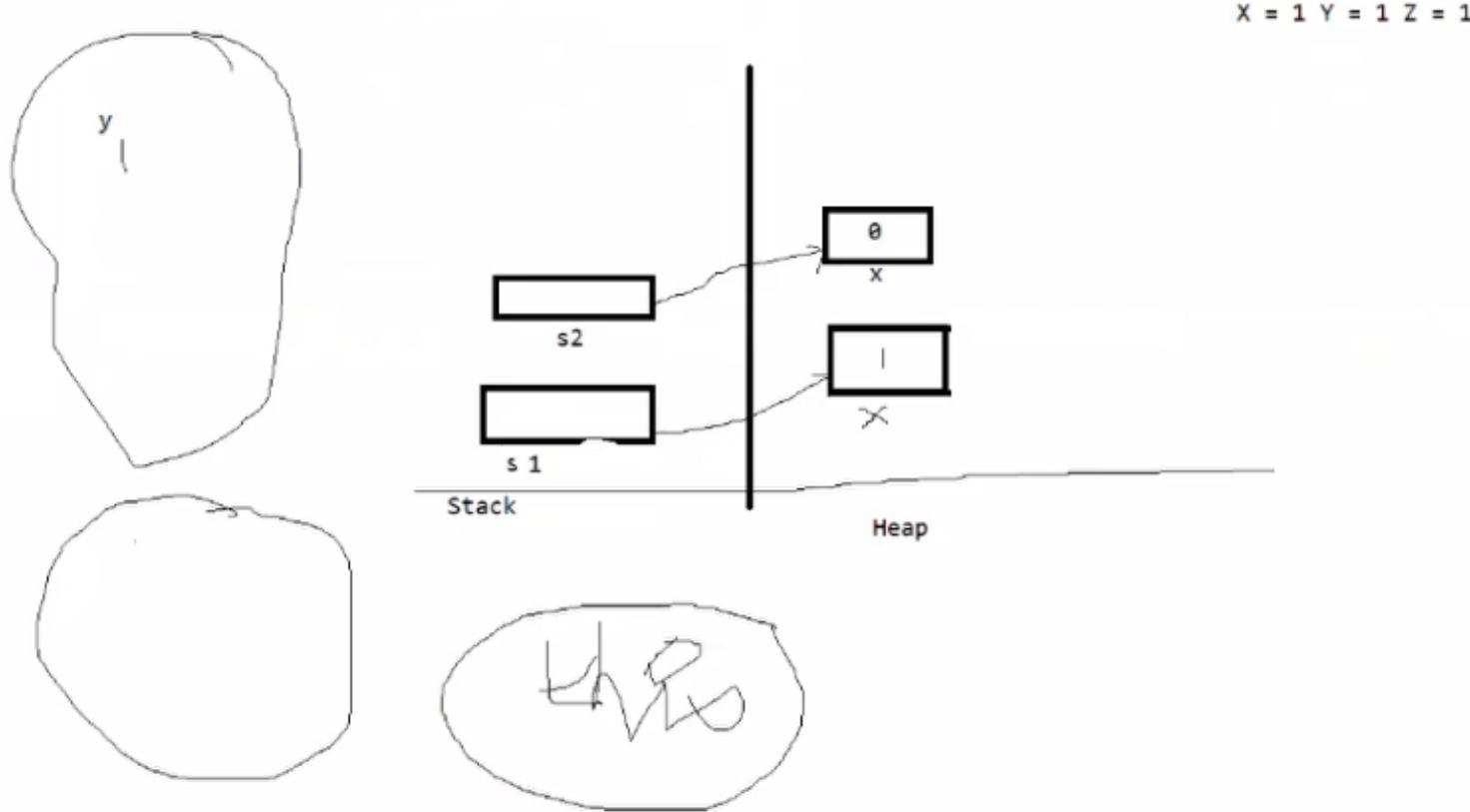
as soon as you say [s2.show](#) the same show method will be called but now it will be pointing to s2 object

So Z is present in method memory and all the methods which are present in the class can be accessed by all the objects

So when i say x++ so x becomes 1, y++ so y becomes 1, z++ z becomes 1 so remember z is a local variable



so now as soon as the program controls comes out  
so z will loose its value



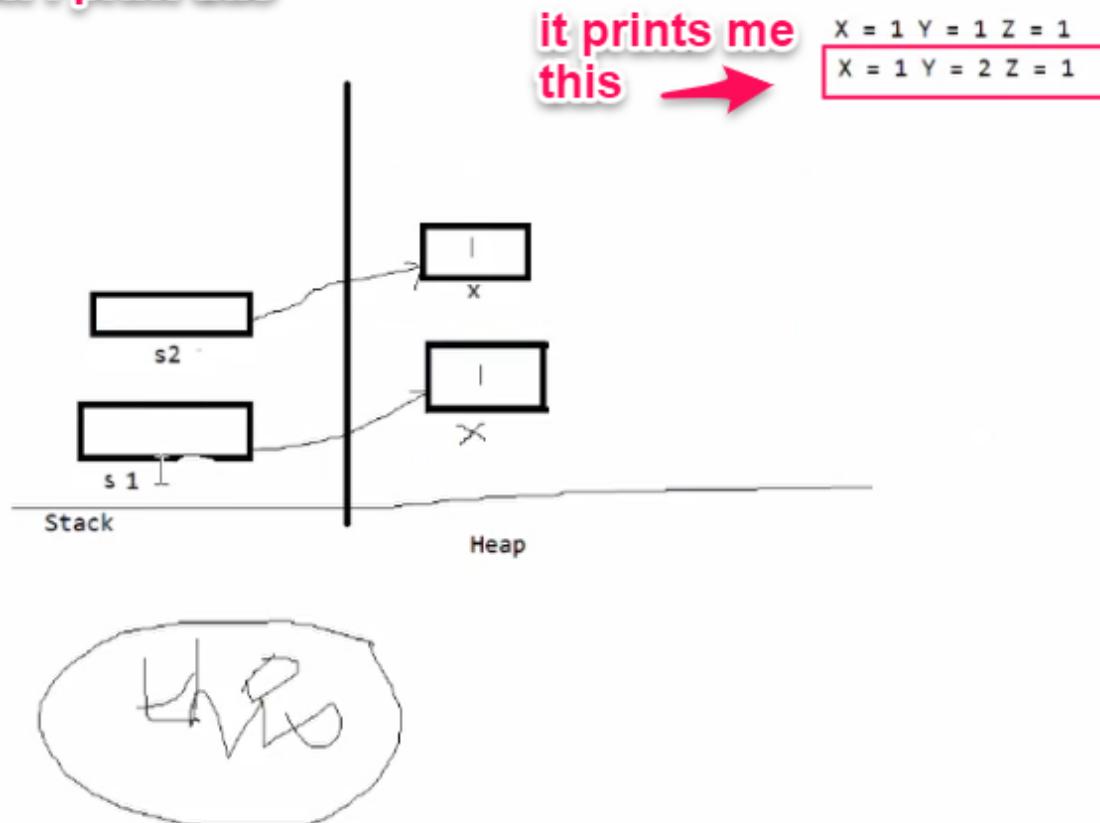
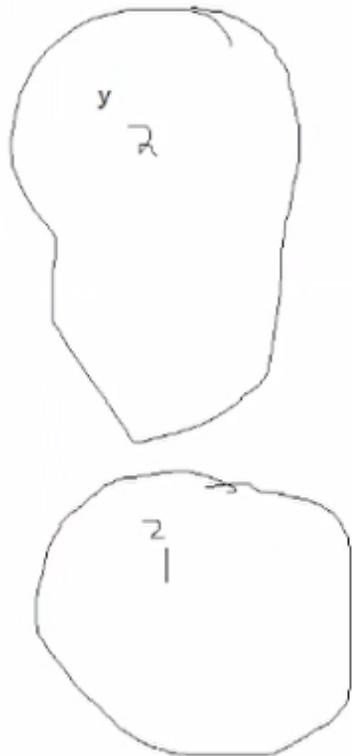
now we say [S2.Show\(\)](#)

so s2 is a different object here now

so x of s2 will becomes 1

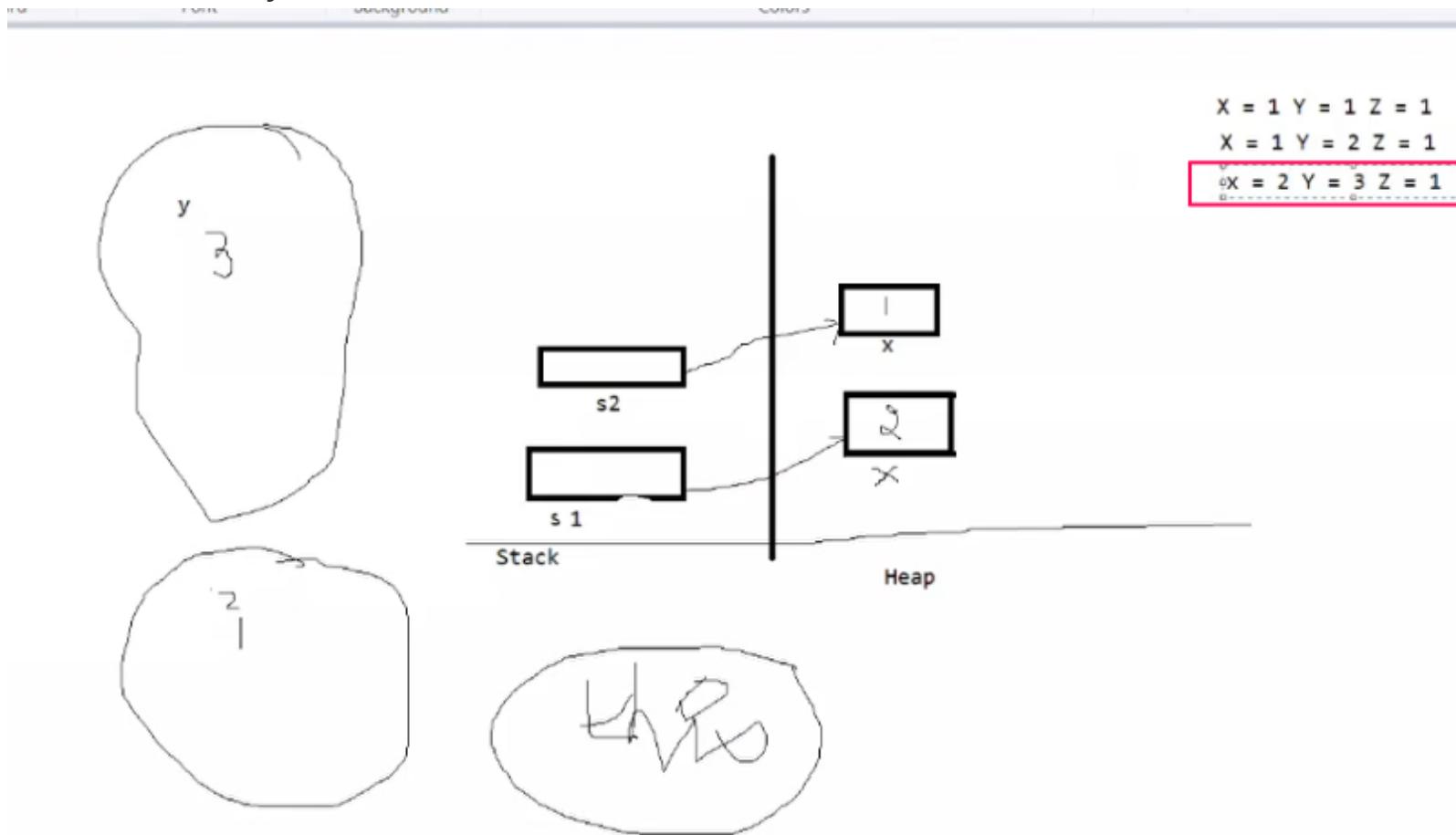
y becomes 2  
and z becomes 1

so again when i print this

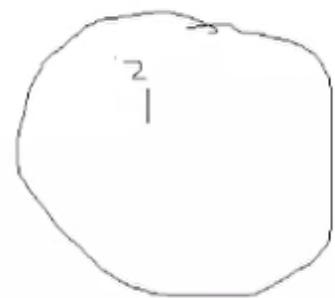
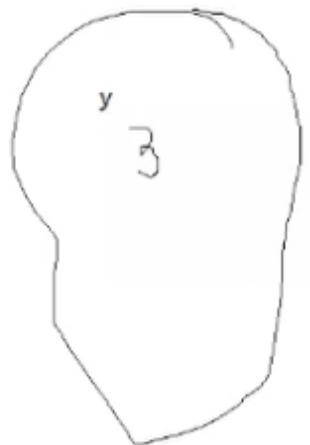


Now s1.show()

x becomes 2, y becomes 3, z becomes 1

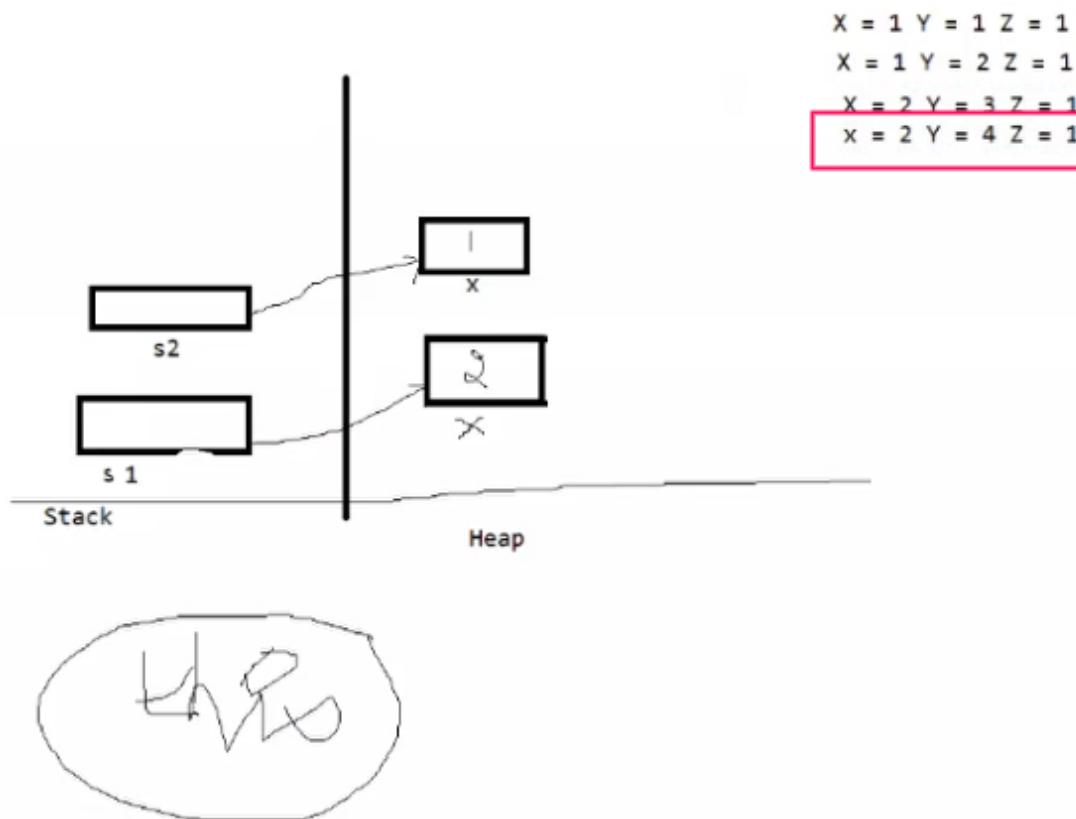


now when i say [S2.Show\(\)](#)

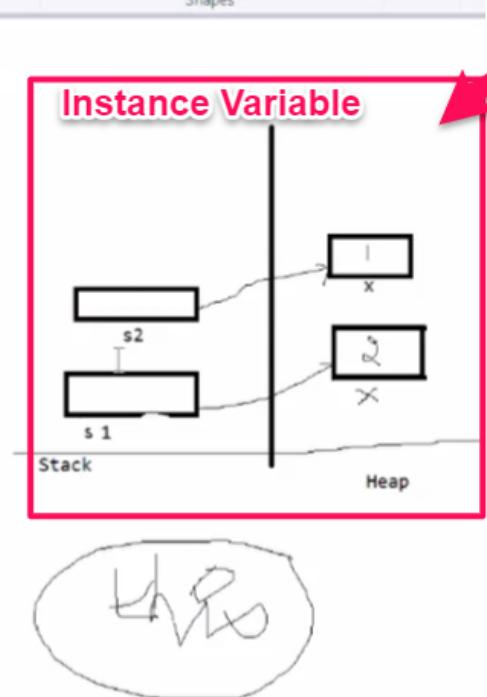
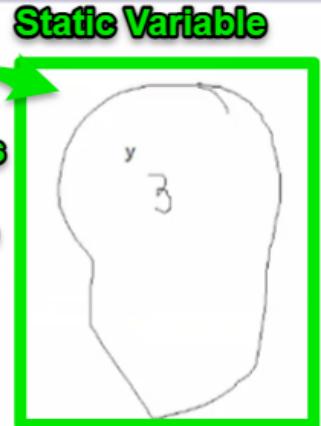


C:\WINDOWS\system32\cmd.exe

```
X = 1    Y = 1    Z = 1
X = 1    Y = 2    Z = 1
X = 2    Y = 3    Z = 1
X = 2    Y = 4    Z = 1
Press any key to continue . . .
```



**Static is like a Water, weather you are there or not static is always there aapke janam ke pehle bhi aapke janam ke baad bhi**



This is you and me

i'll have my own stomach  
you'll have your own stomach

So whenever you say static, so static is which is always loaded in the memory

StaticDemo1.cs   X ConsoleApp\*

ConsoleApp

```
5     using System.Threading.Tasks;
6
7     namespace ConsoleApp
8     {
9         class StaticDemo1
10        {
11            public StaticDemo1()
12            {
13                Console.WriteLine("In Const");
14            }
15
16            static void Dispaly()
17            {
18                Console.WriteLine("In Display");
19            }
20
21            static void Main()
22            {
23                Console.WriteLine("In Main");
24            }
25        }
26    }
```

1 reference  
0 references  
0 references  
0 references

1 reference  
0 references  
0 references

1 reference  
0 references  
0 references

Constructor

this 2 are static blocks

Solution Explorer

- Solution 'ConsoleApp' (2 of 2 projects)
- AccessTest
- ConsoleApp
  - Properties
  - References
  - CollectionsDemo
  - AccessDemo.cs
  - App.config
  - ArrayDemo1.cs
  - ArrayDemo2.cs
  - BreakContinueDemo.cs
  - Calc.cs
  - ClassDemo.cs
  - CommandLineDemo.cs
  - ConvertType.cs
  - EnumDemo.cs
  - GoToDemo.cs
  - IfDemo1.cs
  - IfDemo2.cs
  - IfDemo3.cs
  - InDemo.cs
  - InOutDemo.cs
  - InOutDemo1.cs
  - IntToWord.cs

Diagnostic Tools Properties Git Changes Notifications

121 % No issues found

Output Package Manager Console Error List Immediate Window

Ln: 22 Ch: 31 Col: 40 TABS CRLF

what is this 10 when this 10 gets executed  
when we create instance of this staticDemo1

So my static method is like a water  
It is always there

```
5     using System.Threading.Tasks;
6
7     namespace ConsoleApp
8     {
9         class StaticDemo1
10    {
11        int x = 10;
12        static int y = 10;
13
14        static void Dispaly()
15        {
16            Console.WriteLine("In Display");
17        }
18
19        static void Main()
20        {
21            Console.WriteLine("In Main");
22        }
23    }
24
25
```

```
using System.Threading.Tasks;

namespace ConsoleApp
{
    class StaticDemo1
    {
        int x = 10;
        static int y = 10;

        static void Dispaly()
        {
            Console.WriteLine("In Display");
        }

        static void Main()
        {
            Console.WriteLine("In Main");
            StaticDemo1 s1 = new StaticDemo1();
        }
    }
}
```

and i get leg and hand only when i created  
so these variable what we have created  
here, will alive only when  
we have instance of that  
Created Other wise this x is useless but this y is always there

So when ever you create instance  
it like when i take birth so  
while creation only i'll have  
my hand leg every thing

So when i take birth the water is already there and i'm  
trying to consume it

StaticDemo1.cs\* X ConsoleApp\*

ConsoleApp

```
5     using System.Threading.Tasks;
6
7     namespace ConsoleApp
8     {
9         class StaticDemo1
10        {
11            int x = 10;
12            static int y = 10;
13
14            static void Dispaly()
15            {
16                y = 100;
17                Console.WriteLine("In Display");
18            }
19
20            static void Main()
21            {
22                Console.WriteLine("In Main");
23                StaticDemo1 s1 = new StaticDemo1();
24            }
25        }
    }
```

121 % No issues found

Output Package Manager Console Error List ... Immediate Window

Can i access Y here is it possible



The screenshot shows a code editor with a file named `StaticDemo1.cs`. The code defines a `ConsoleApp` namespace containing a `StaticDemo1` class. The class has two static integer fields: `x` and `y`, both initialized to 10. It also contains a `Display()` method that changes the value of `y` to 100 and prints "In Display" to the console. The `Main()` method creates an instance of `StaticDemo1` and prints "In Main". The code editor interface includes tabs for `StaticDemo1.cs` and `ConsoleApp*`, a status bar showing "121 %", and a message "No issues found".

```
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class StaticDemo1
10     {
11         int x = 10;
12         static int y = 10;
13
14         static void Dispaly()
15         {
16             y = 100;
17             Console.WriteLine("In Display");
18         }
19
20         static void Main()
21         {
22             Console.WriteLine("In Main");
23             StaticDemo1 s1 = new StaticDemo1();
24         }
25     }
26 }
```

I have a salt water

And

I have a River Water

Both are water both are always there

i'm taking little bit of salt water  
and adding it to river water

Is this Possible?  
YES it is Possible

```
StaticDemo1.cs*  # X ConsoleApp*
```

```
ConsoleApp
```

```
ConsoleApp.StaticDemo1
```

```
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class StaticDemo1
10     {
11         int x = 10;
12         static int y = 10;
13
14         static void Dispaly()
15         {
16             y = 100;
17             x = 100; // Red arrow points here
18             Console.WriteLine("In Display");
19         }
20
21         static void Main()
22         {
23             Console.WriteLine("In Main");
24             StaticDemo1 s1 = new StaticDemo1();
25         }
26     }
27 }
```

```
Output Package Manager Console Error List ... Immediate Window
```

```
eady
```

Can i do this ?

No, because x is a instance variable  
and x will get life only when you create instance  
but water is always there

So static cannot access the element present inside me  
water cannot access me

So static is something which is automatically gets created  
and has nothing to do with instance

So thats why the instance variable cannt be accessed  
inside the static methods

```
StaticDemo1.cs*  X  ConsoleApp*
ConsoleApp  ConsoleApp.StaticDemo1

8   {
9     2 references
10    class StaticDemo1
11    {
12      int x = 10;           Instance Variable
13      static int y = 10;    Class Variable
14      0 references
15      void Show()
16      {
17        x = 100;
18      }
19      0 references
20      static void Dispaly()
21      {
22        y = 100;
23        //x = 100;
24        Console.WriteLine("In Display");
25      }
26      0 references
27      static void Main()
28      {
```

121% No issues found

Output Package Manager Console Error List Immediate Window

ready

Type here to search

Instance Variable

Class Variable

Instance Method

this will be created only  
when you create object

Class Method

When ever you put static  
it means it belongs to class  
Meaning all the instances can  
be accessible

StaticDemo1.cs\* ✘ × ConsoleApp\*

ConsoleApp

ConsoleApp.StaticDemo1

```
13
14     void Show()      //instance method
15     {
16         x = 100;
17     }
18
19     static StackOverflowException
20     {
21         STAThreadAttribute
22         StaticDemo
23         StaticDemo1
24         String
25         StringBuilder
26         StringComparer
27         StringComparison
28         Show
29         Console.WriteLine("In Main");
30         StaticDemo1 s1 = new StaticDemo1();
31     }
32
33 }
```

1 reference

0 references

0 references

0 references

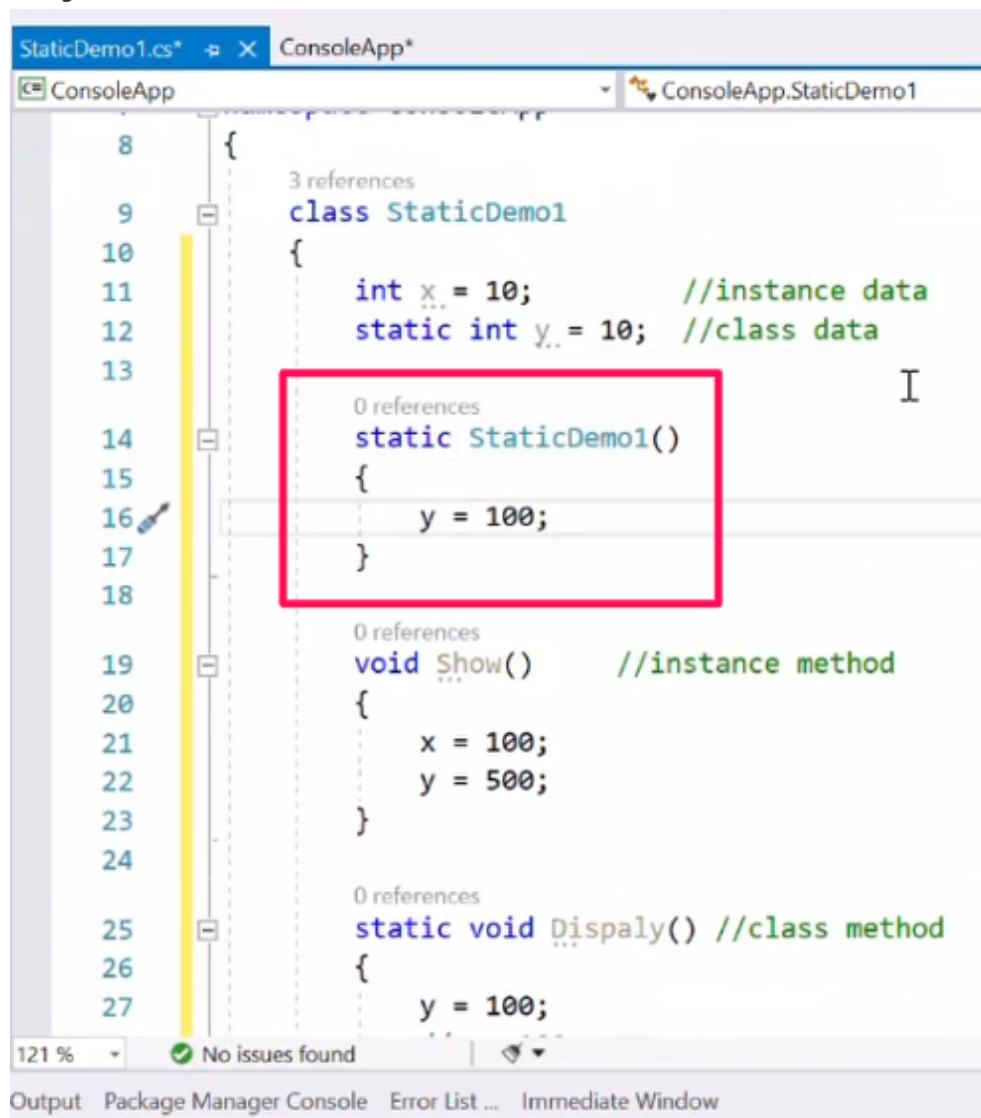
Look at this  
you cannot directly say  
Show();

```
13
14     void Show()      //instance method
15     {
16         x = 100;
17     }
18
19     static void Dispaly() //class method
20     {
21         y = 100;
22         //x = 100;
23         Console.WriteLine("In Display");
24     }
25
26     static void Main()
27     {
28         Dispaly
29         Console.WriteLine("In Main");
30         StaticDemo1 s1 = new StaticDemo1();
31     }
32
33 }
```

But you can directly access display

So i have a static member here and i want to initialize them how can I Initialize ?

so just create a static ctor (constructor)



```
StaticDemo1.cs*  X  ConsoleApp*
ConsoleApp          ↴ ConsoleApp.StaticDemo1

8      {
9          3 references
10         class StaticDemo1
11             int x.. = 10;           //instance data
12             static int y.. = 10;   //class data
13
14             0 references
15             static StaticDemo1()
16                 {
17                     y = 100;
18
19             0 references
20             void Show()           //instance method
21                 {
22                     x = 100;
23                     y = 500;
24
25             0 references
26             static void Dispaly() //class method
27                 {
28                     y = 100;

121 %  No issues found
```

The screenshot shows the code editor for a C# file named StaticDemo1.cs. The code defines a class StaticDemo1 with instance variables x and y, and class variables x and y. It includes a static constructor that initializes y to 100. The static constructor is highlighted with a red box. The code editor interface includes tabs for StaticDemo1.cs and ConsoleApp\*, and a status bar at the bottom.

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** StaticDemo1.cs > X ConsoleApp
- Toolbars:** Standard, Task List, Tools, Help
- Code Editor:** The code is written in C# and defines a class named `StaticDemo1`.
  - Line 13: `static StaticDemo1()` - 0 references.
  - Line 14: `{`
  - Line 15: `y = 100;`
  - Line 16: `Console.WriteLine("In static const");`
  - Line 17: `}`
  - Line 19:  (empty line)
  - Line 20: `public StaticDemo1()` - 2 references.
  - Line 21: `{`
  - Line 22: `Console.WriteLine("In Const");`
  - Line 23: `}`
  - Line 24:  (empty line)
  - Line 25: `static void Main()` - 0 references.
  - Line 26: `{`
  - Line 27: `Console.WriteLine("In Main");`
  - Line 28: `StaticDemo1 s1 = new StaticDemo1();`
  - Line 29: `StaticDemo1 s2 = new StaticDemo1();` - A yellow lightbulb icon is placed here, indicating a warning or suggestion.
  - Line 30: `}`
  - Line 31: `}`
  - Line 32: `}`
  - Line 33:  (empty line)
- Status Bar:** 121 04

What will be the output ?

```
C:\WINDOWS\system32\cmd.exe
In static const
In Main
In Const
In Const
Press any key to continue . . .
```

you can see that i'm creating multiple instances

and even though i have multiple instances the static constructor will be called only once  
and look at this it is called even before main is called

So Meaning if you create any information as static so whether you create a object or  
not the static information is already there

**Assignment - Write a Program to Auto Generate Student Id,  
Accept 5 students name and print student id and student Name**

# ReadOnly Variable And Constant

## Declaring Readonly Variables and Constants

### Constant Variable

compile time

- Value of constant field is obtained at compile time
- Value is stored in Assembly (in Type Metadata section).
- No memory space allocation during runtime
- Can be assigned only through field-initialization technique

- Value of read-only field is obtained at run time
- Value is not stored in assembly.
- Memory space is allocated during run time
- Can be assigned through constructor and field-initialization technique



runtime

Read-only variable

the data of these cannot be modified

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class ReadOnlyDemo
10     {
11         const int x = 10;
12         readonly int y = 10;
13     }
14 }
15
16
17
```

```
ReadOnlyDemo.cs*  X  ConsoleApp*
ConsoleApp          ConsoleApp.R

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class ReadOnlyDemo
10     {
11         const int x = 10;
12         readonly int y = 10;
13
14         void Test()
15         {
16             x++;
17             y++;
18         }
19     }
20 }
21

121 %  ②  0  ← →  ⌂
```

Can you see i'm getting error

Now Remember Constants are not variables we cannot, never call constant as variable

because the word itself says vary meaning we can change the value  
and in constants we cannot modify or change the value

**We cannot modify the value of Constant and Readonly**



Look at this it says left hand side of a assignment must be variable, property or indexer  
it should be a variable meaning x is not a variable so you cannot initialize them



But where as y dosen't have any error, i can still put value in y

**Meaning whenever you create a constant and you don't initialize you get error**

**but where as readonly i'm not initializing i'll not get the error**

**So while declaring you should and must initialize the constant**

ReadOnlyDemo.cs\* X ConsoleApp\*

ConsoleApp

ConsoleApp.ReadOnlyC

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class ReadOnlyDemo
10     {
11         const int x; ←
12         readonly int y;
13
14         public ReadOnlyDemo()
15         {
16             y = 100;
17         }
18
19         void Test()
20         {
21             x++;
22         }
23     }
24 }
```

1 reference

0 references

0 references

Output Package Manager Console Error List ... Immediate Window

Type here to search

