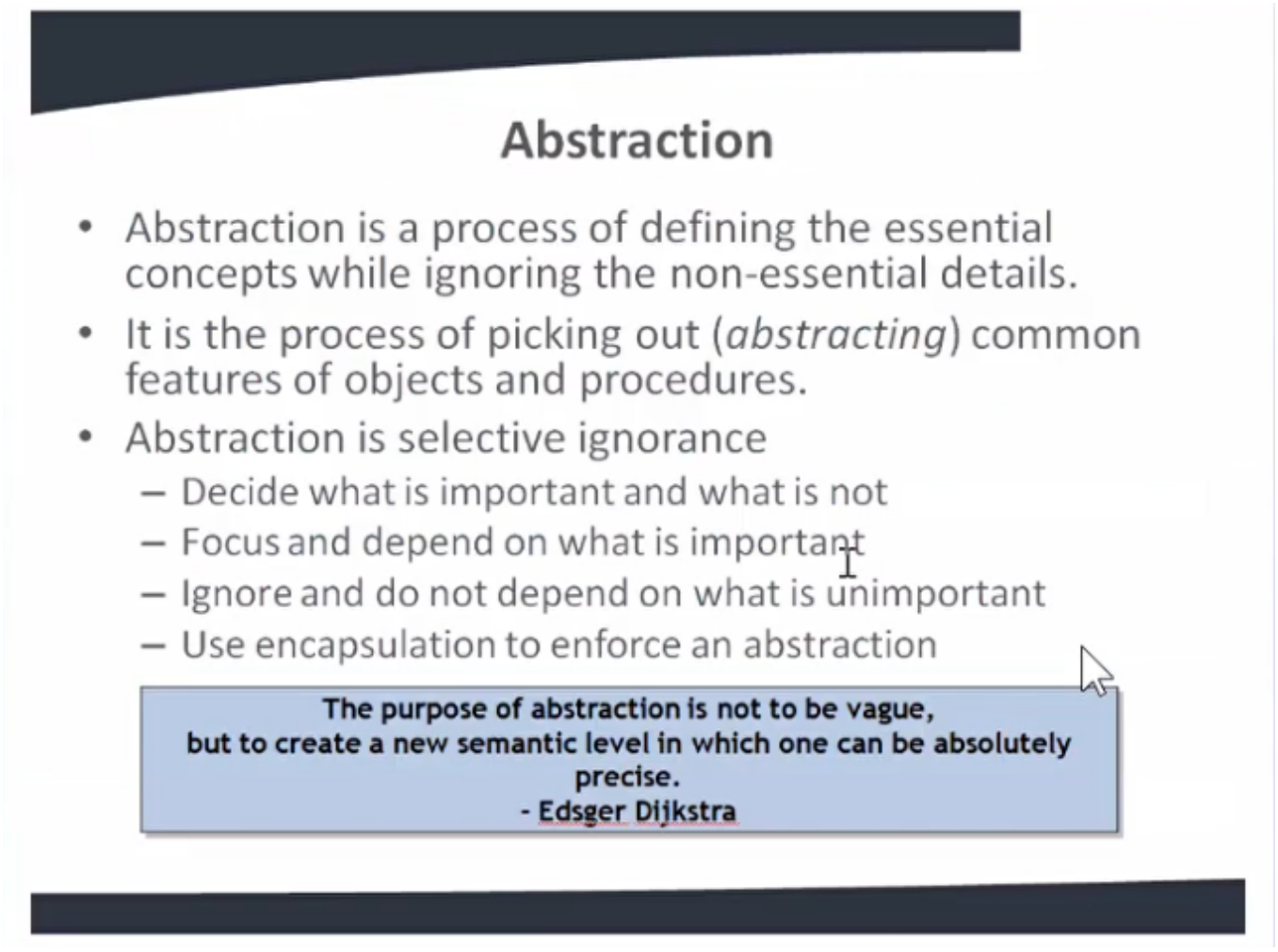
C#-8

Abstraction



Its a Internal Implementation which you are trying to show to the external person

SO the Information which you are trying to expose/Show

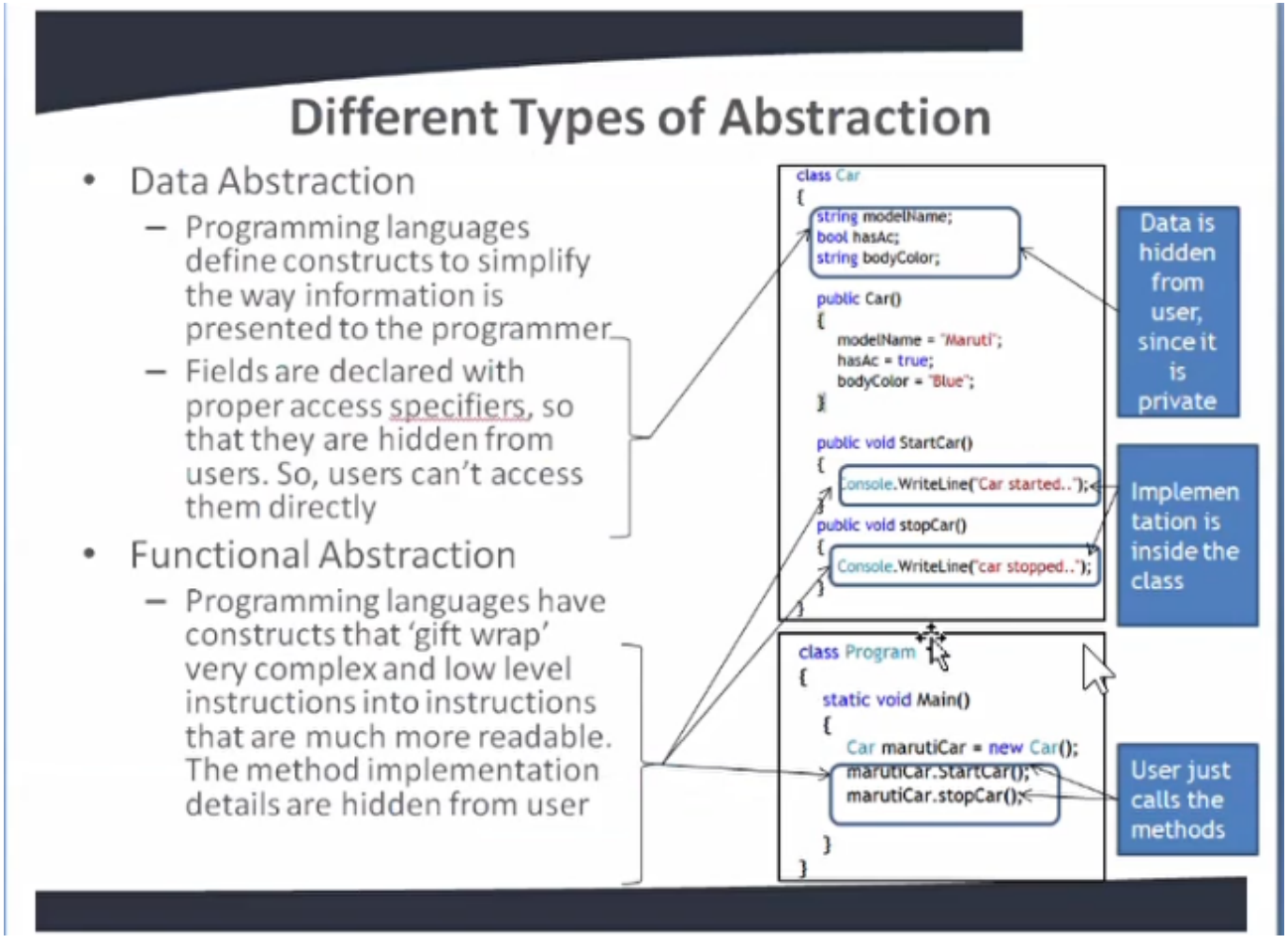
Hiding the implementation Showing the Feature of a particular object

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



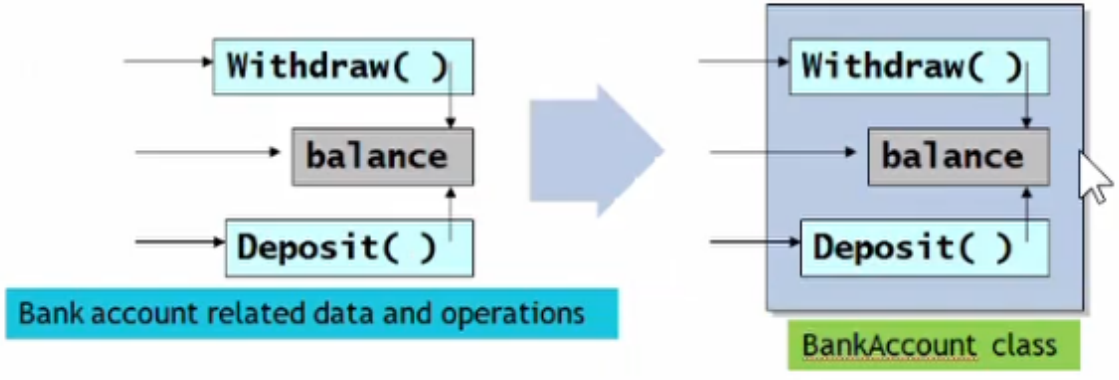
If We want our data members to be hidden this will be my data abstraction

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?

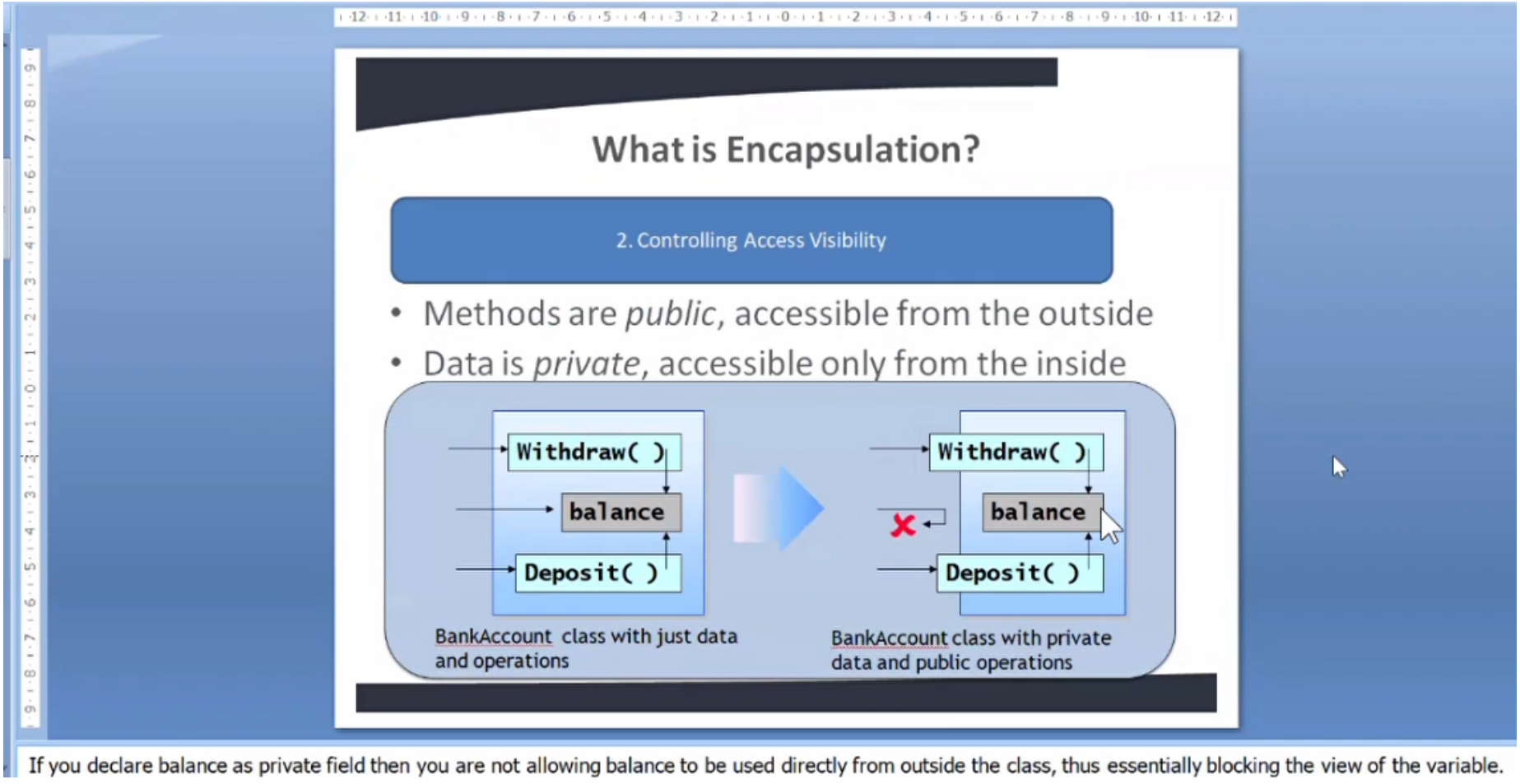
1. Combining Data and Methods

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

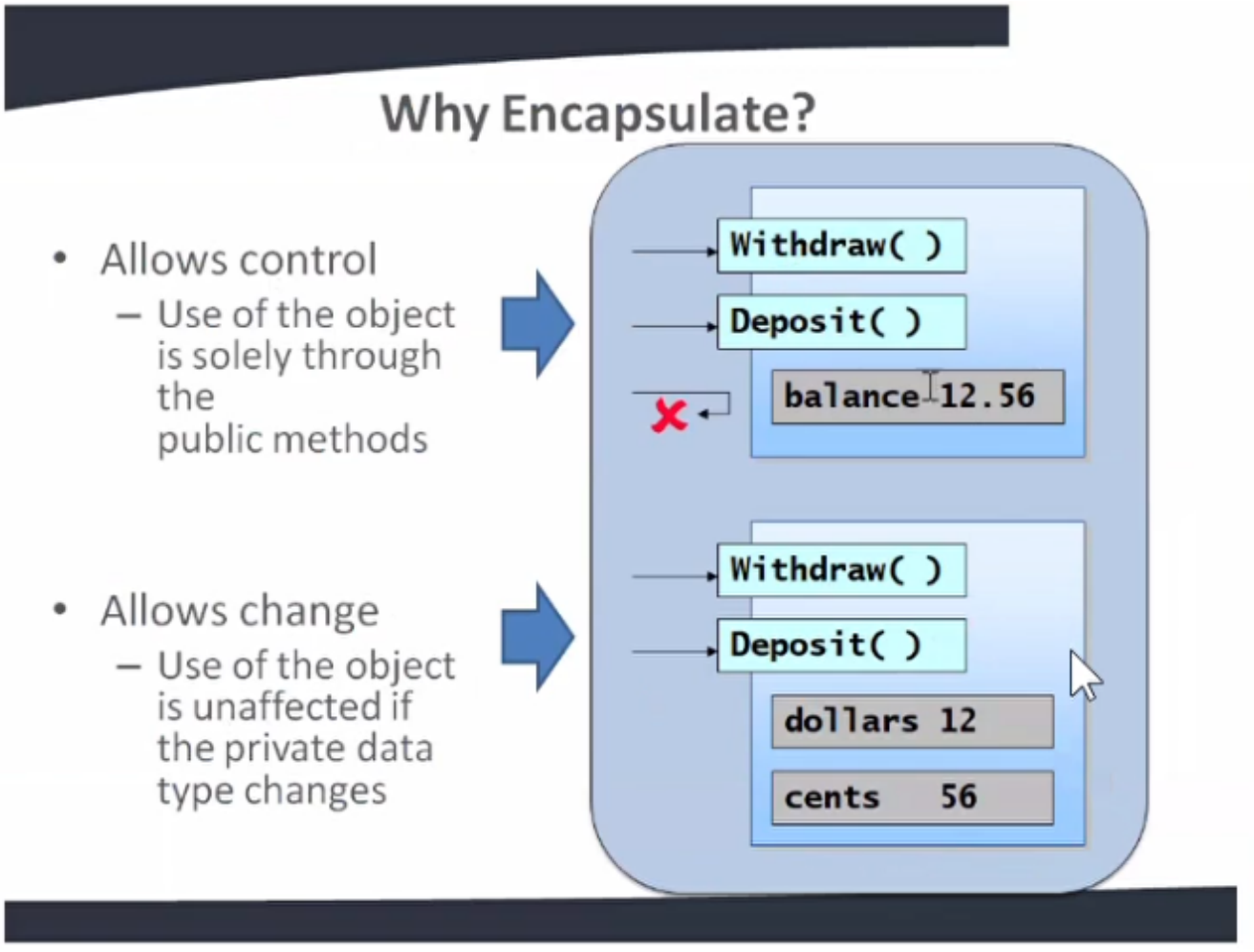


Consider a bank account, where account maintains a balance and also supports behavior such as, withdrawal and deposit of money.

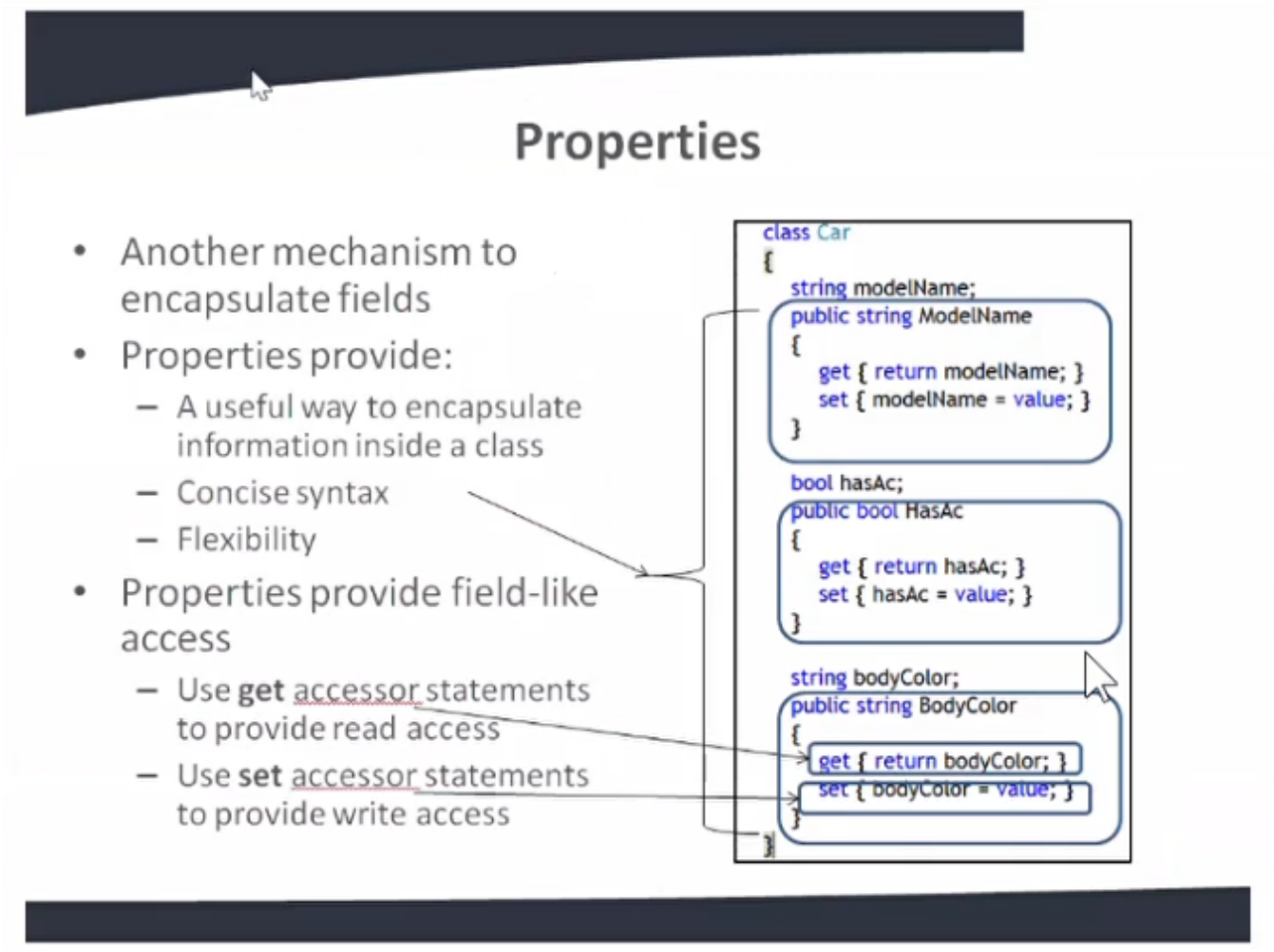
there will be lot of abstractions which we can write for example
withdraw, deposite i can expose but Balance i can always Encapsulate
withdraw, deposite i can expose to the user but whereas balance i can hide it
So there will be calculate method and the withdraw and deposite method will use this calculate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



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



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



We have already discussed property tags

There is **no memory allocation for a property**

How to use Property?

- Set value of private fields though 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 g string 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 cannot 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
readonly	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

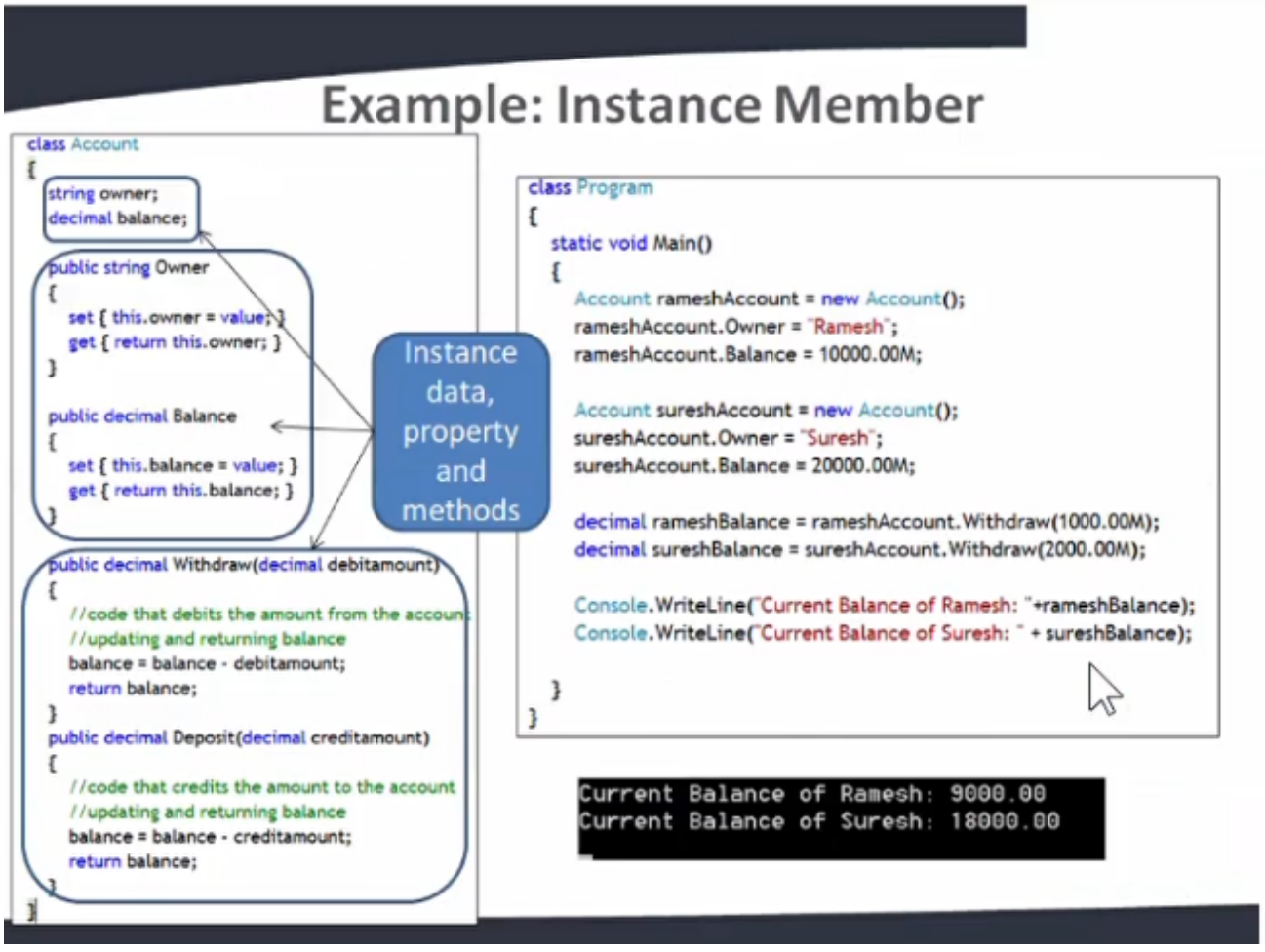
Class and Class Member Access Specifiers

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

```
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; }
    }
}
```

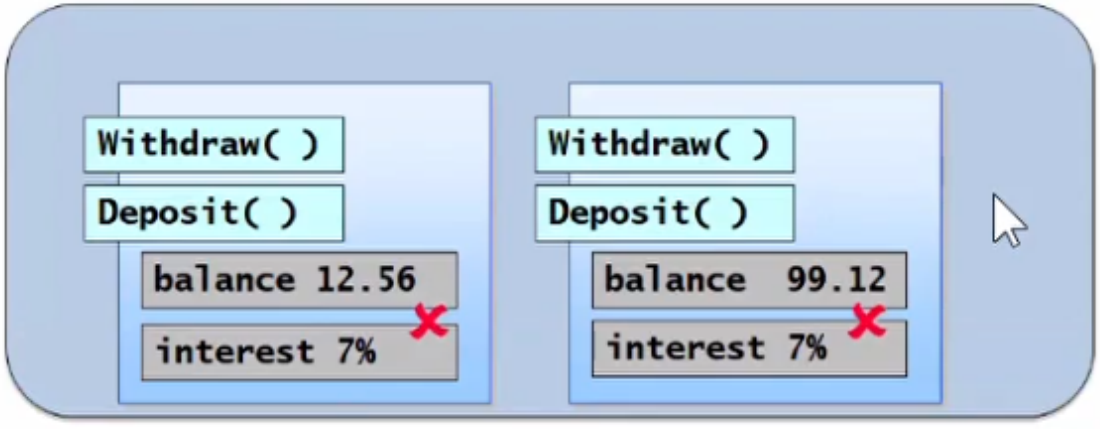
If this statement were modelname = modelname; What would happen?

```
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;
    }
}
```

Static Members

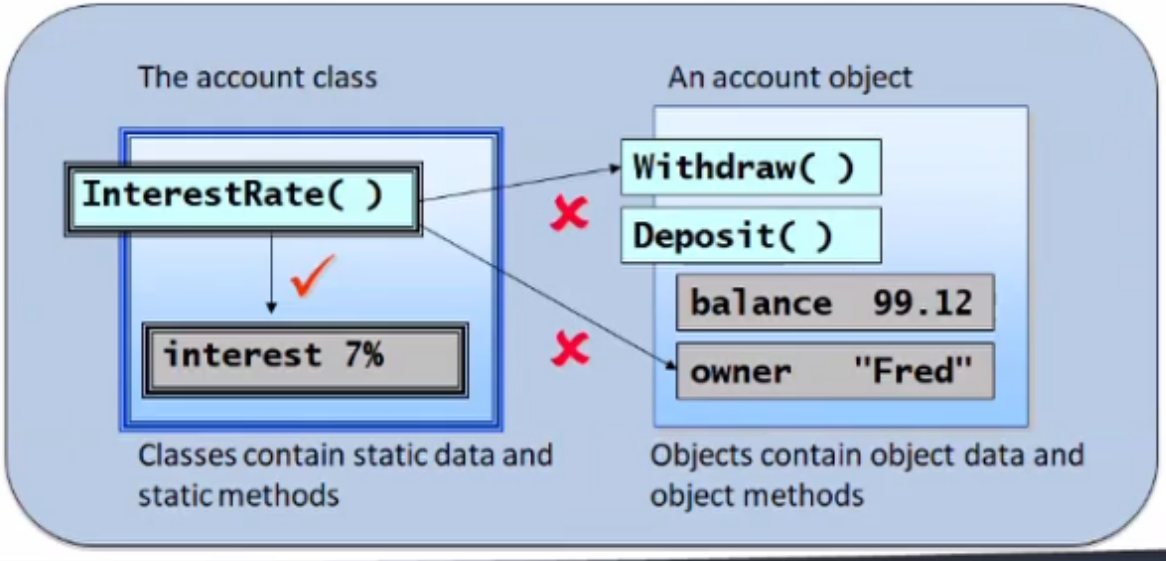
Using Static Data

- 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

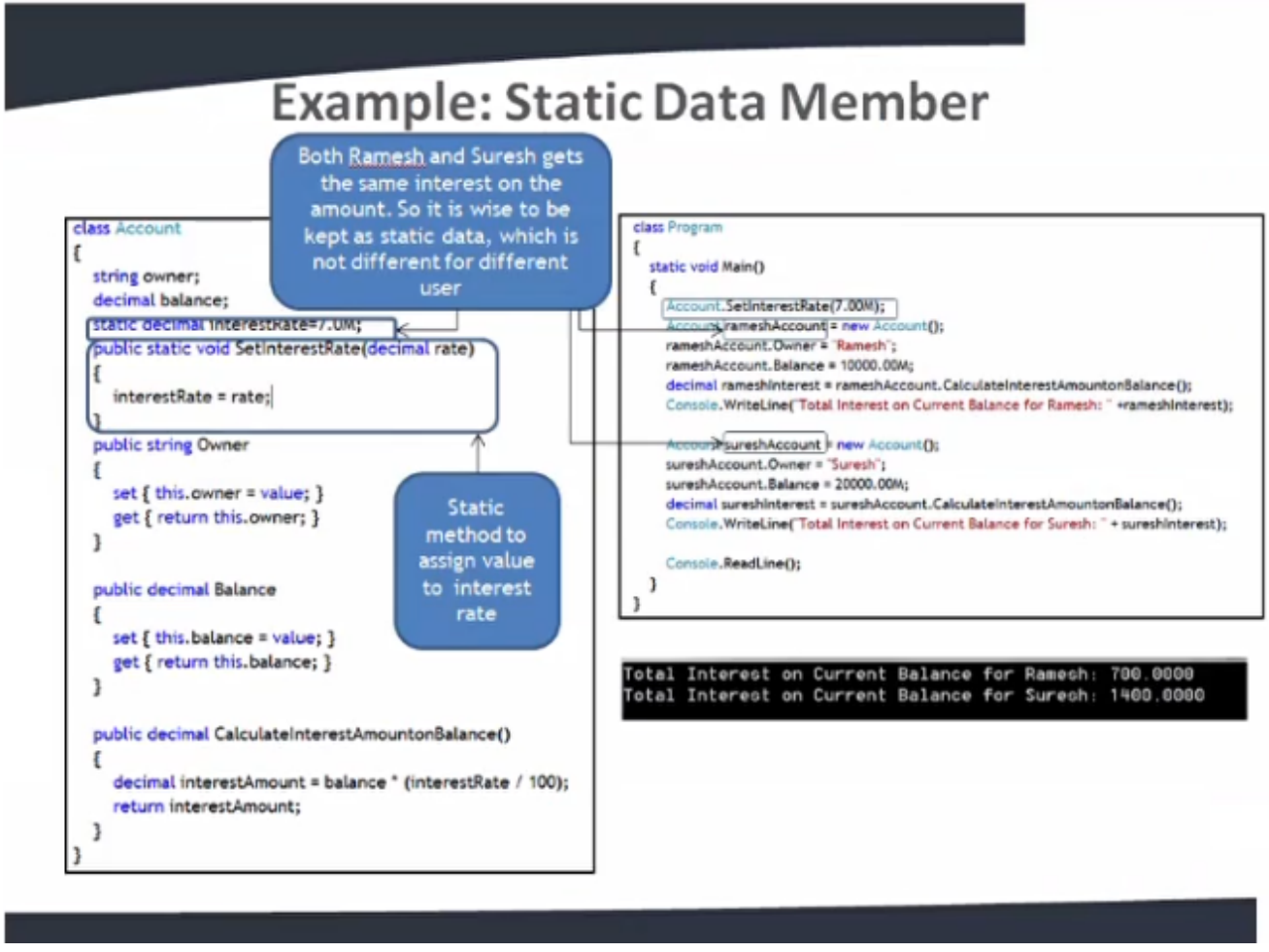


Using Static Methods

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



Example: Static Data Member




```
class Account
{
   string owner;
   decimal balance;
   static decimal interestRate=7.0M;
   public static void SetInterestRate(decimal rate)
   {
      interestRate = rate;
   }
   public string Owner
   {
      set { this.owner = value; }
      get { return this.owner; }
   }

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

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

```
class Program
{
   static void Main()
   {
      Account.SetInterestRate(7.00M);
      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();
   }
}
```

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

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:

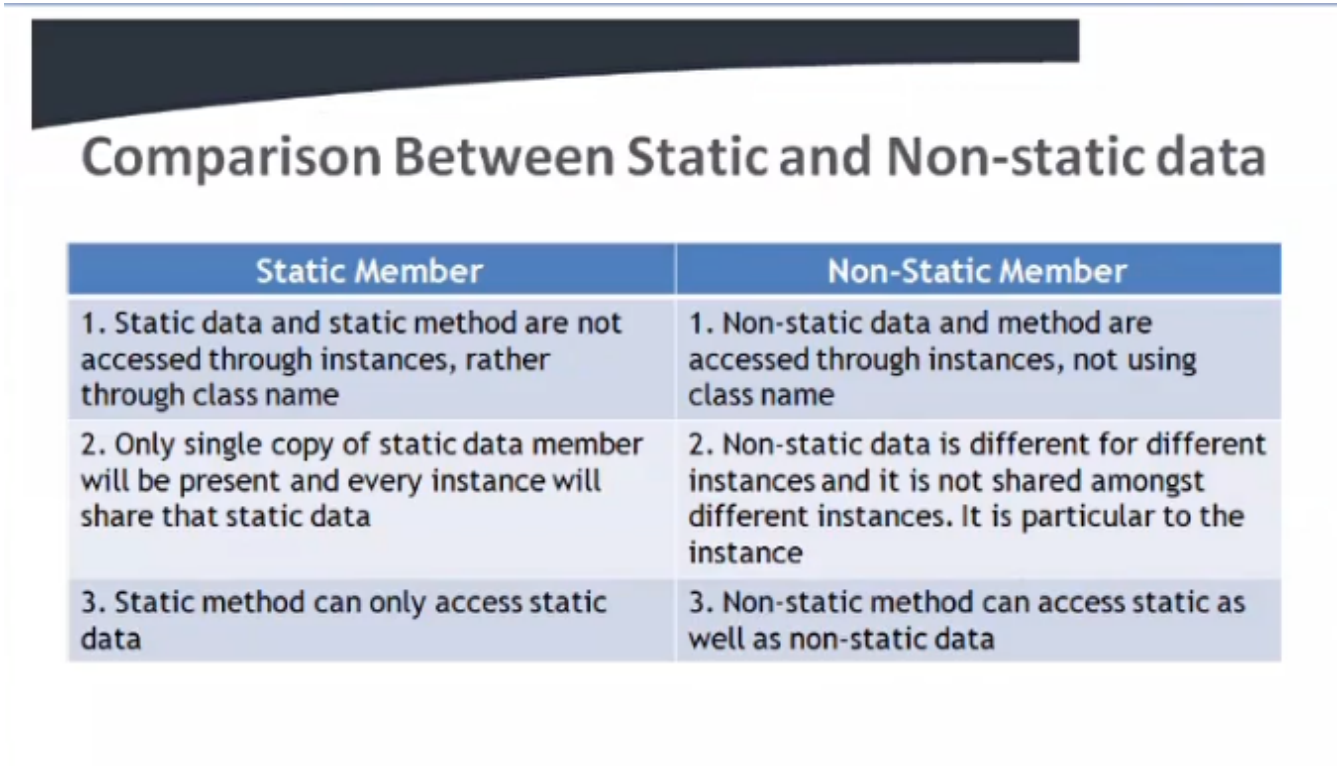
```
class Account
{
   string owner;
   decimal balance;
   static decimal interestRate;

   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

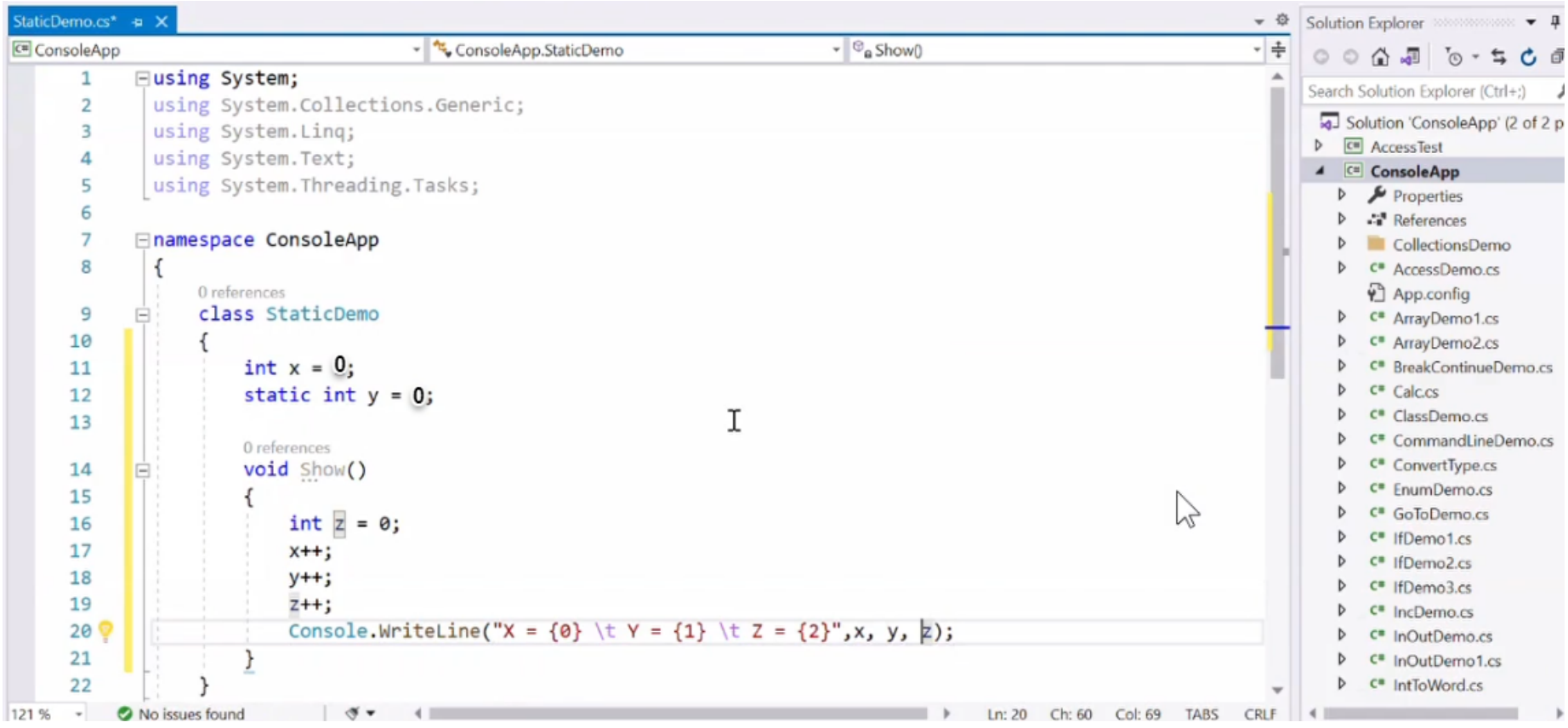
```
class Program
{
   static void Main(string[] args)
   {
      Account rameshAccount = new Account();
      rameshAccount.Owner = "Ramesh";
      rameshAccount.Balance = 10000.00M;
      decimal rameshInterest = rameshAccount.CalculateInterestAmountonBalance();
      Console.WriteLine("Total Interest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

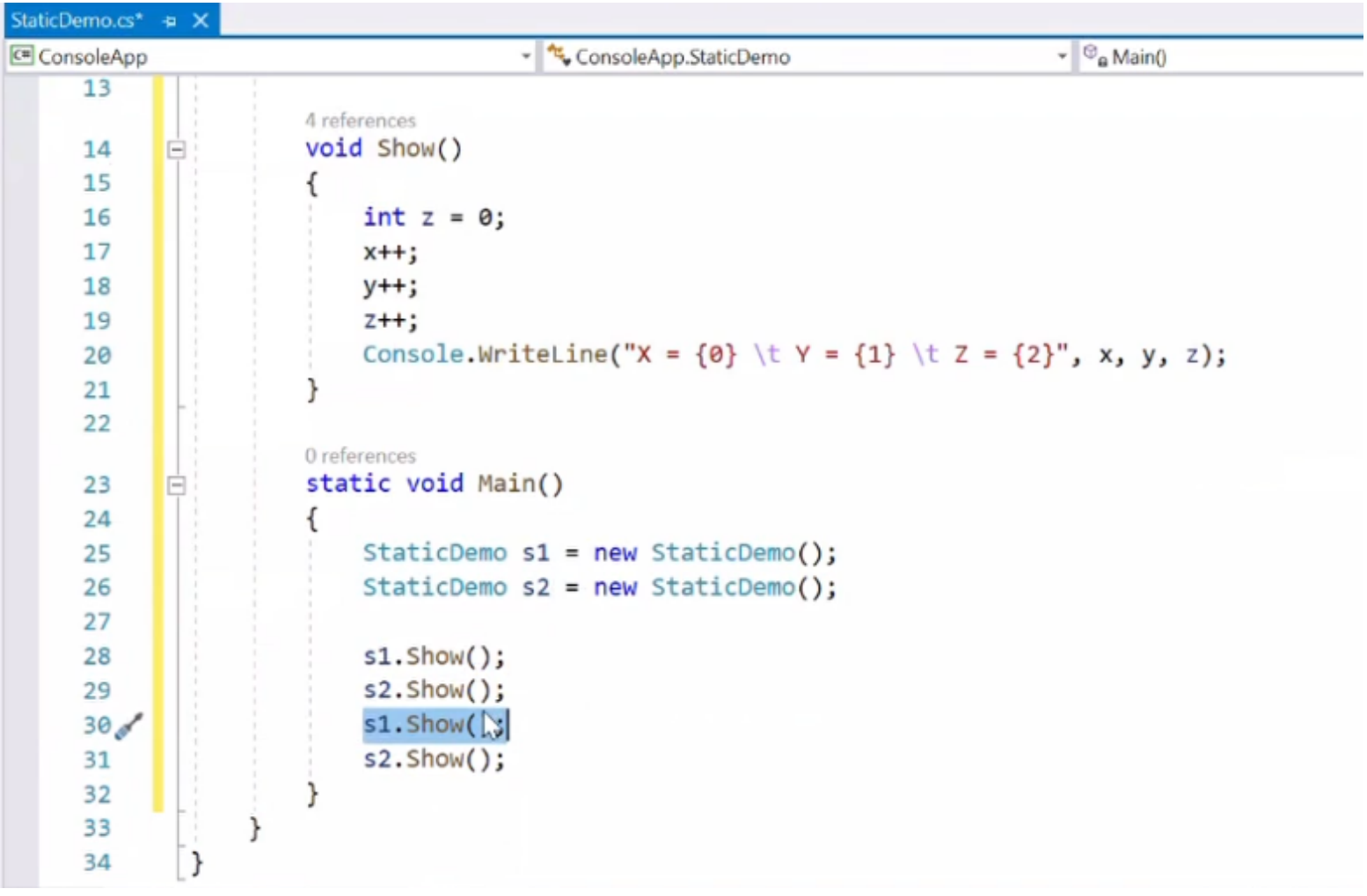
```
Interest rate is: 7.0
Total Interest on Current Balance for Ramesh: 700.0000
Total Interest on Current Balance for Suresh: 1400.0000
```

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



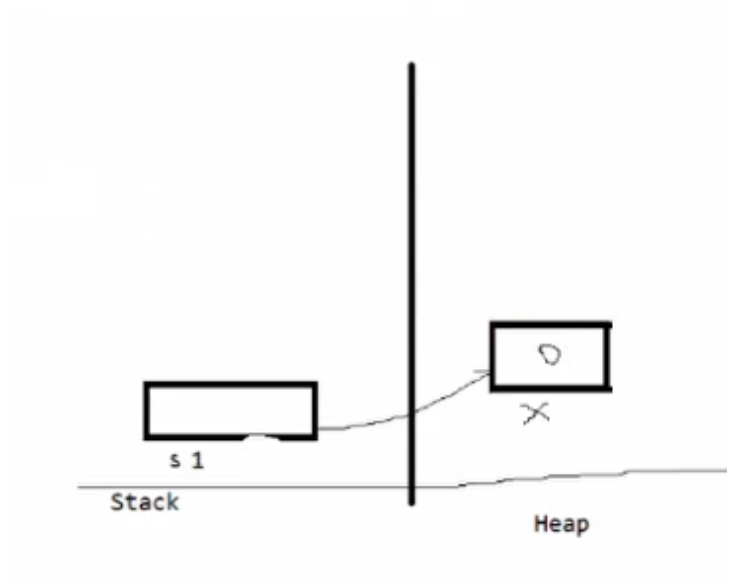
What are static members


StaticDemo.cs*
ConsoleApp
ConsoleApp.StaticDemo
Show()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
0 references
class StaticDemo
{
int x = 0;
static int y = 0;
0 references
void Show()
{
int z = 0;
x++;
y++;
z++;
Console.WriteLine("X = {0} \t Y = {1} \t Z = {2}",x, y, z);
}
}
121 %
No issues found
Ln: 20
Ch: 60
Col: 69
TABS
CRLF
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApp' (2 of 2 p
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



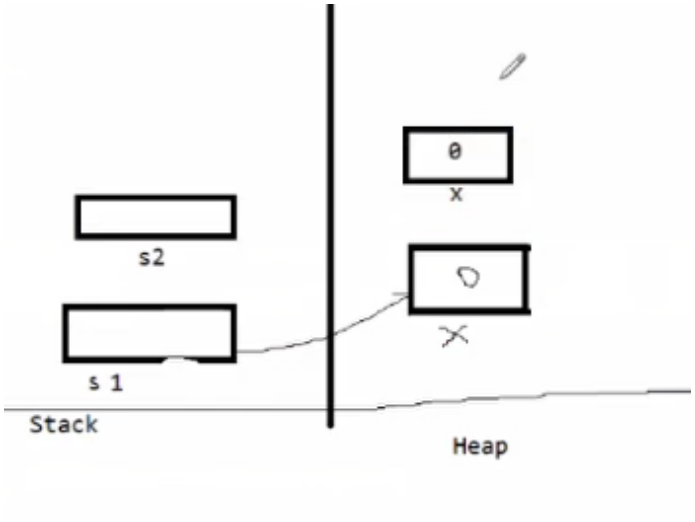
What will be the output of this program ?

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

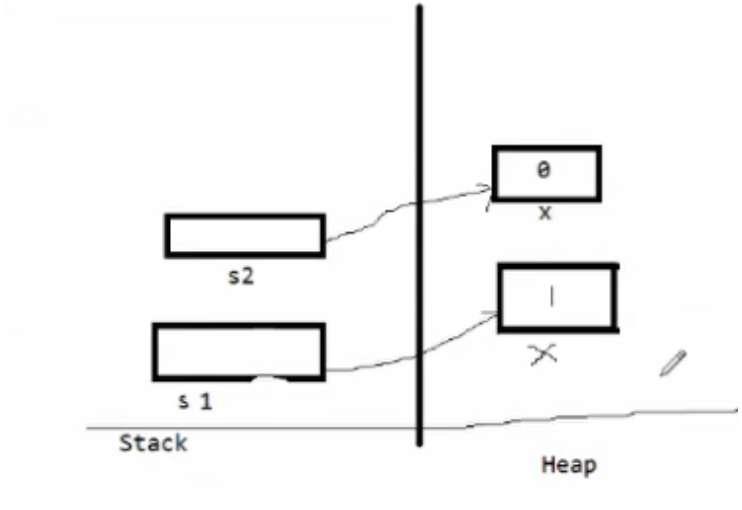


this is what happen in line no 25

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



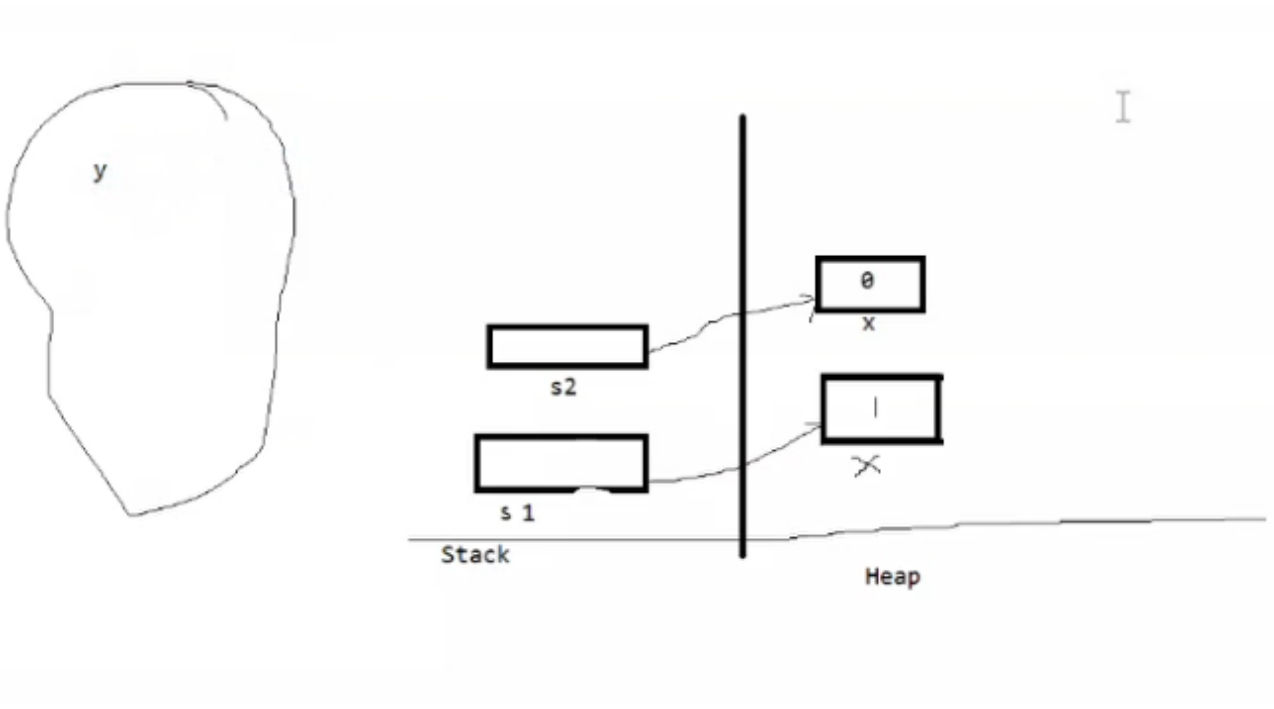
Now when you say x1.display, so it will say x++ so x value of this becomes 1

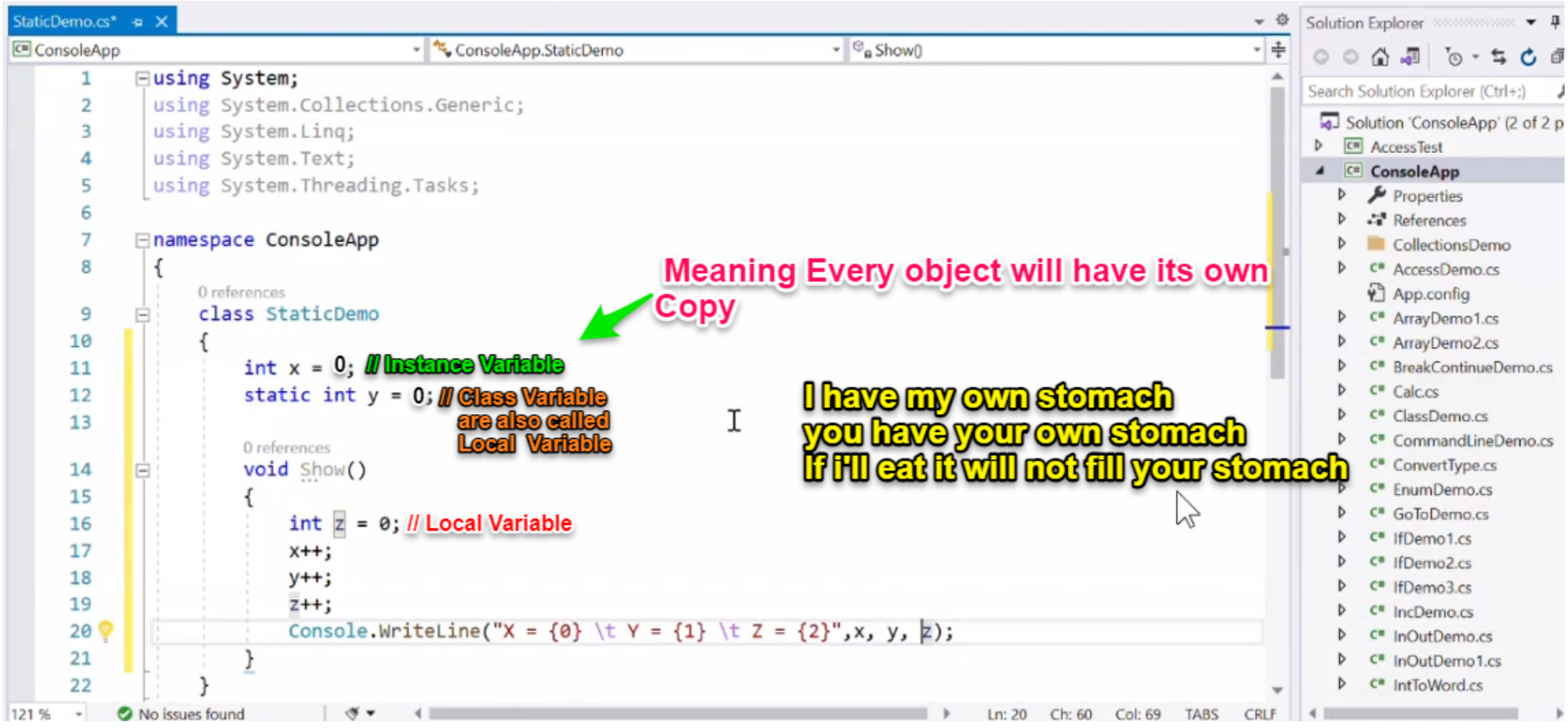


Where is y then ?

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

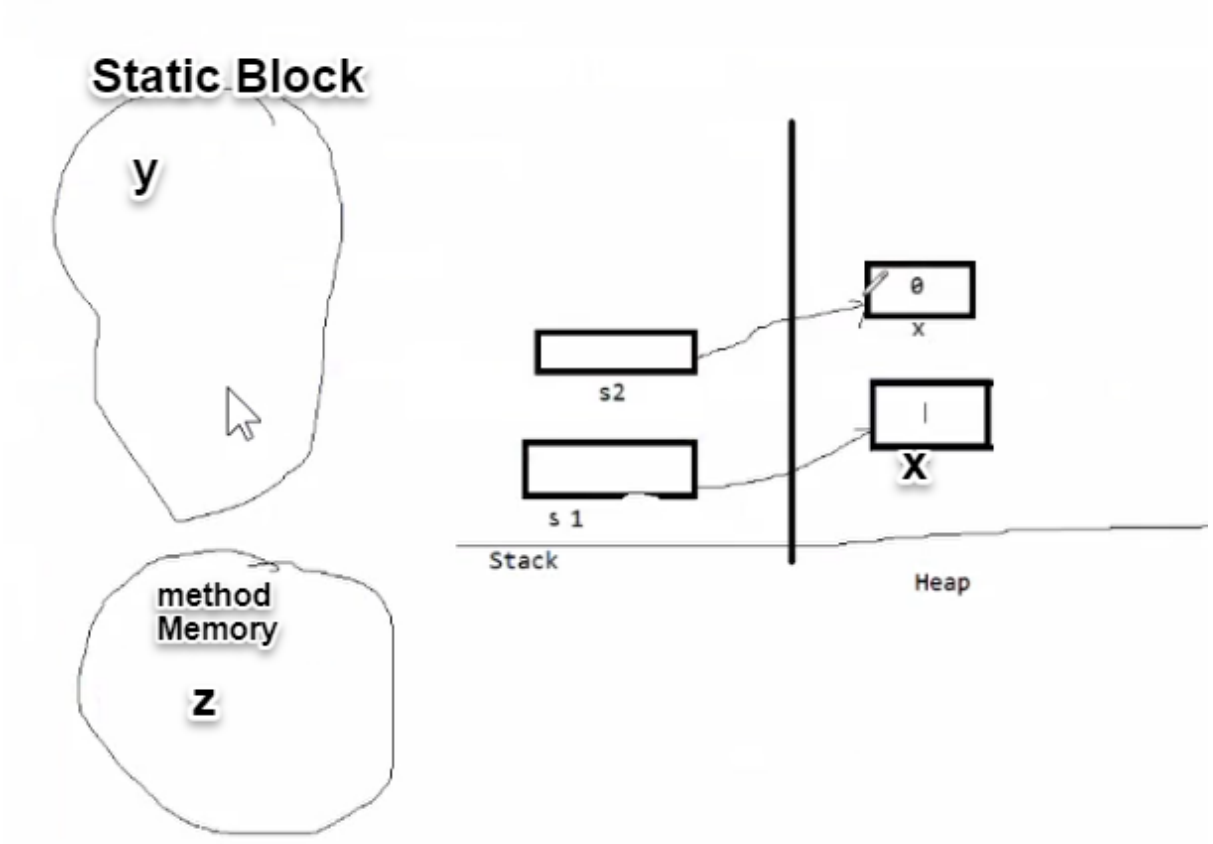

y
0
x
s2
1
s 1
Stack
Heap



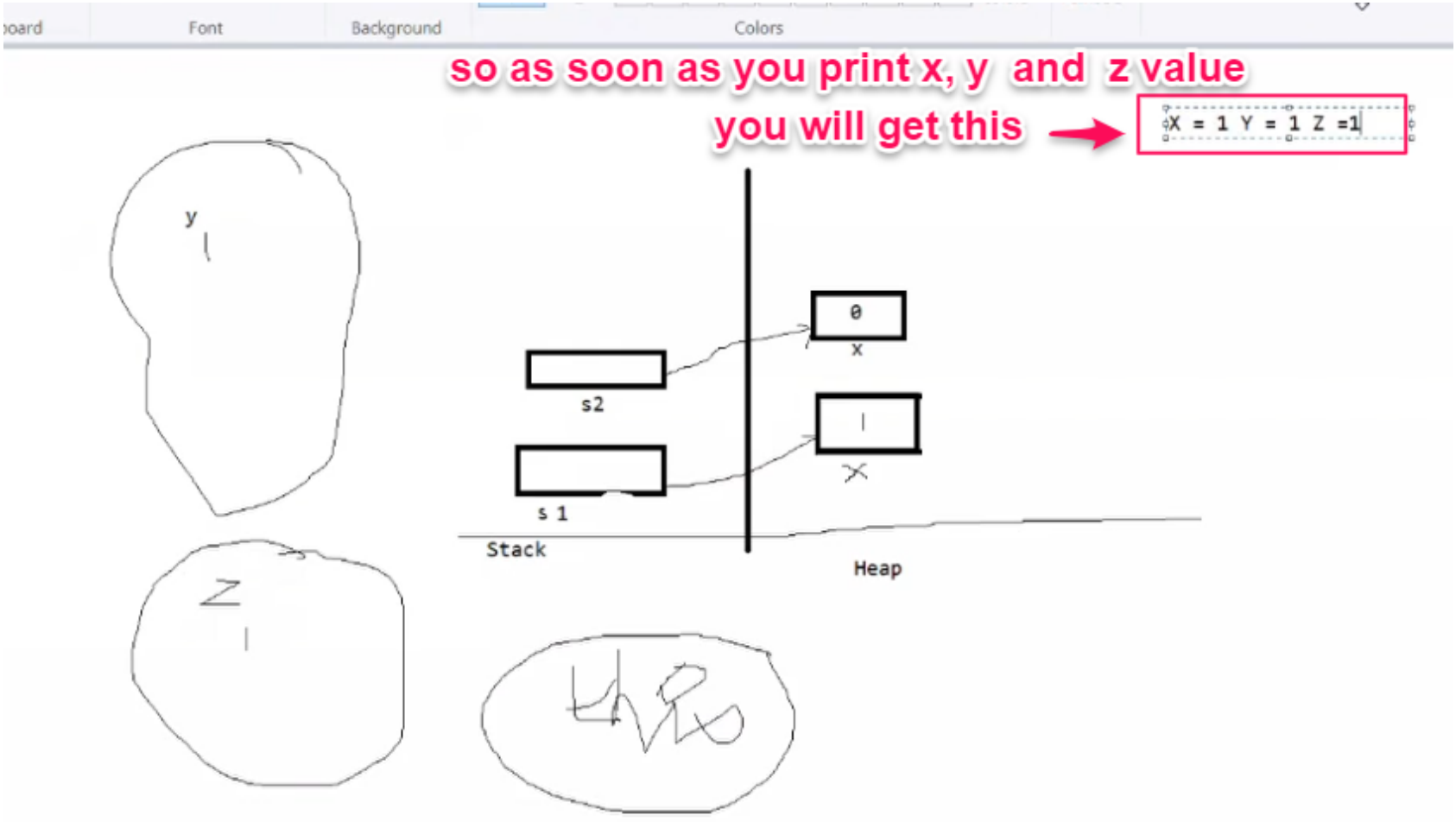
Z will be present with the method, you'll have method memory

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*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

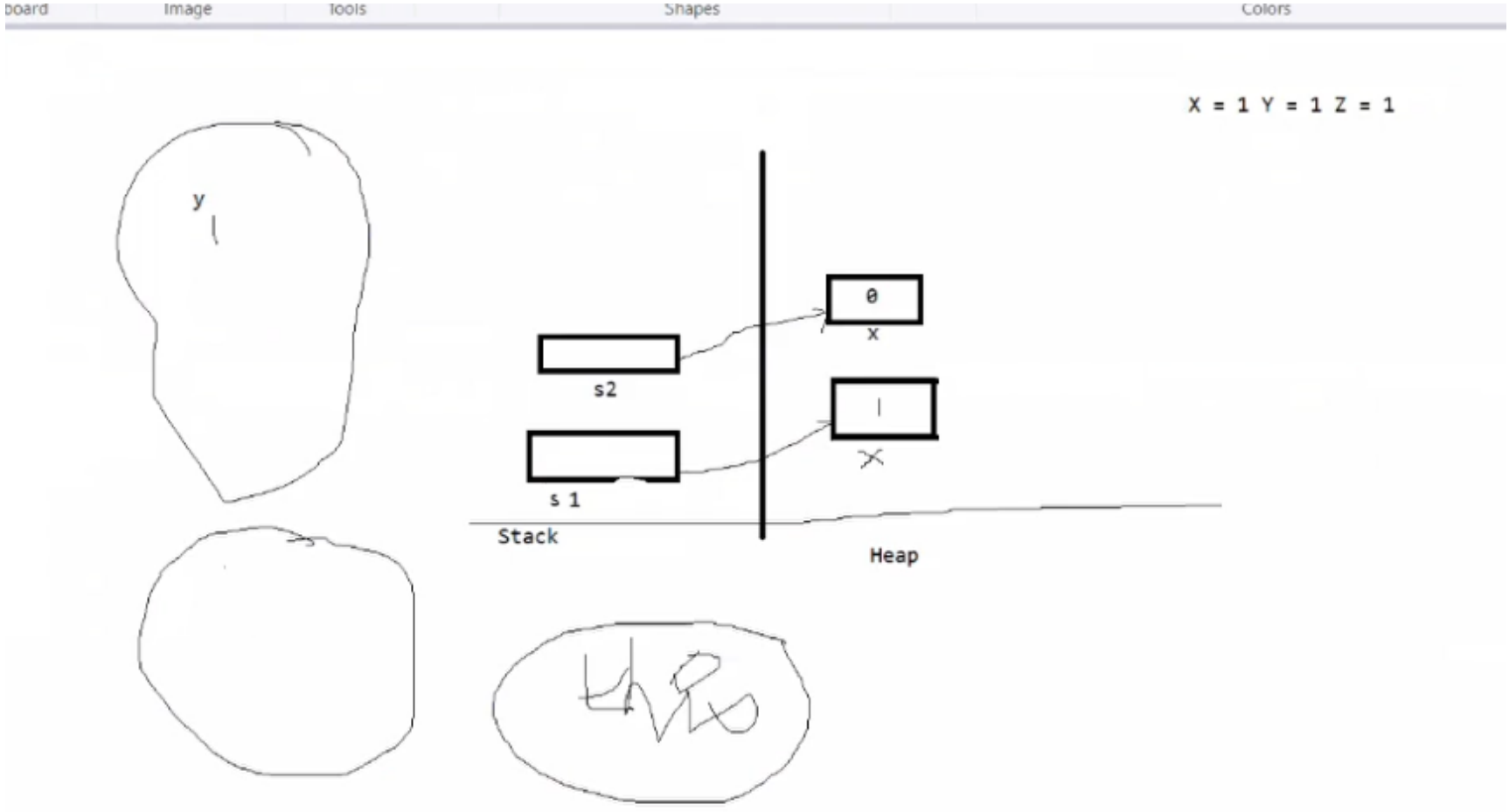


so you have a method memory and whenever this object is passed when you call s1.show() internally it will pass **this** and this will be tagged with that particular object, so your operation system is smart enough it knows which method has been called or CLR will know which method is been called
So this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

so z will loose its value



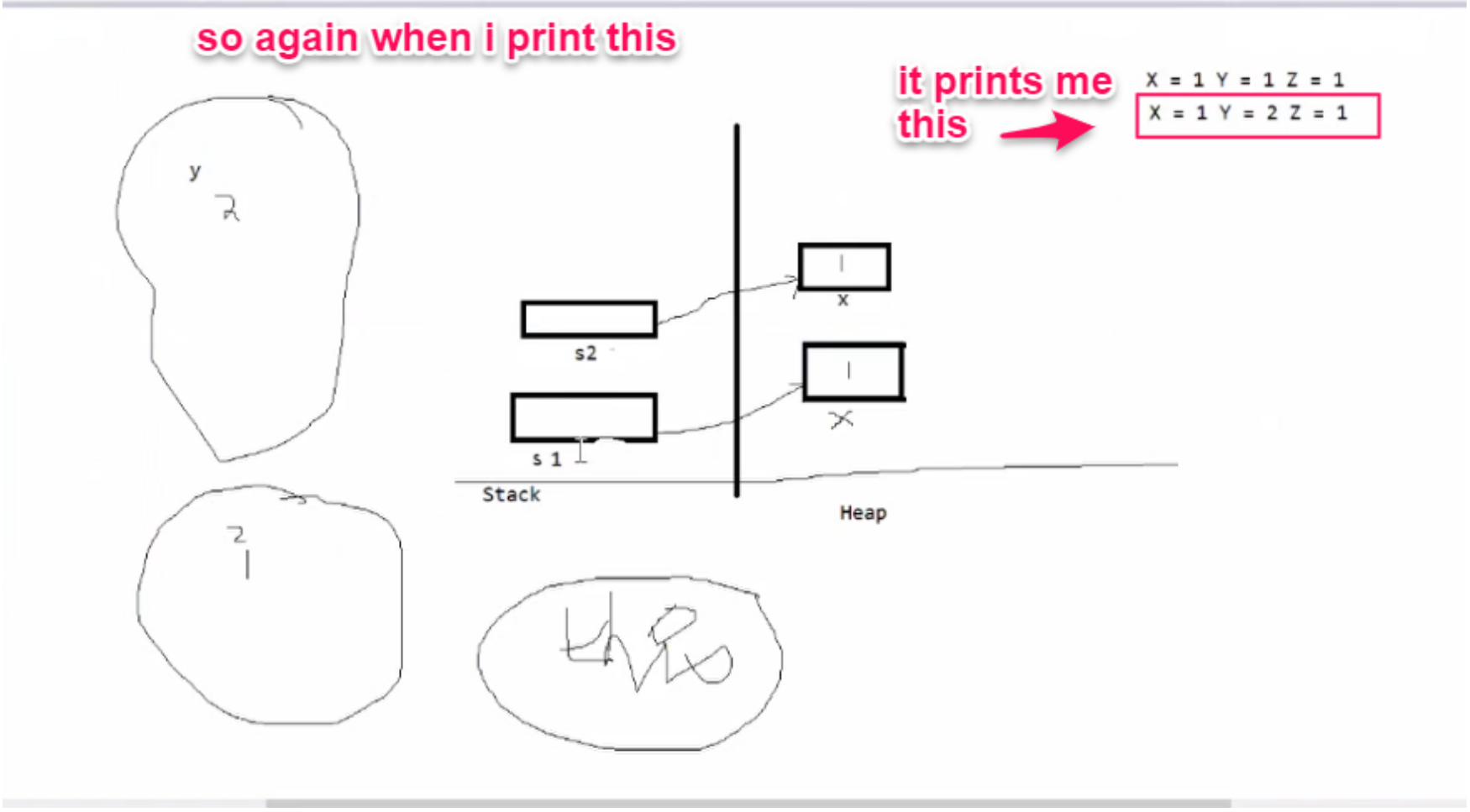
now we say S2.Show()

so s2 is a different object here now

so x of s2 will becomes 1

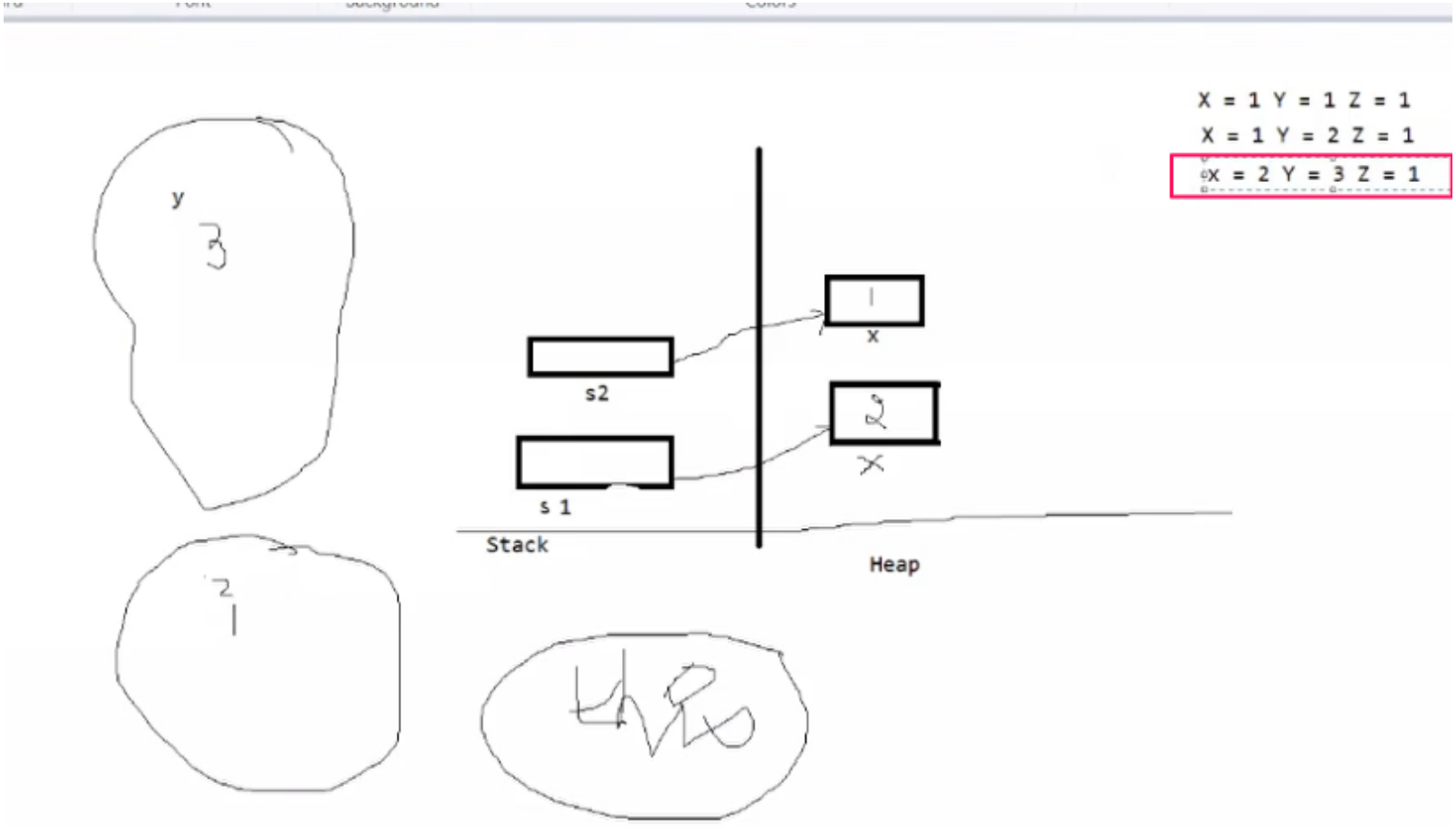
y becomes 2

and z becomes 1



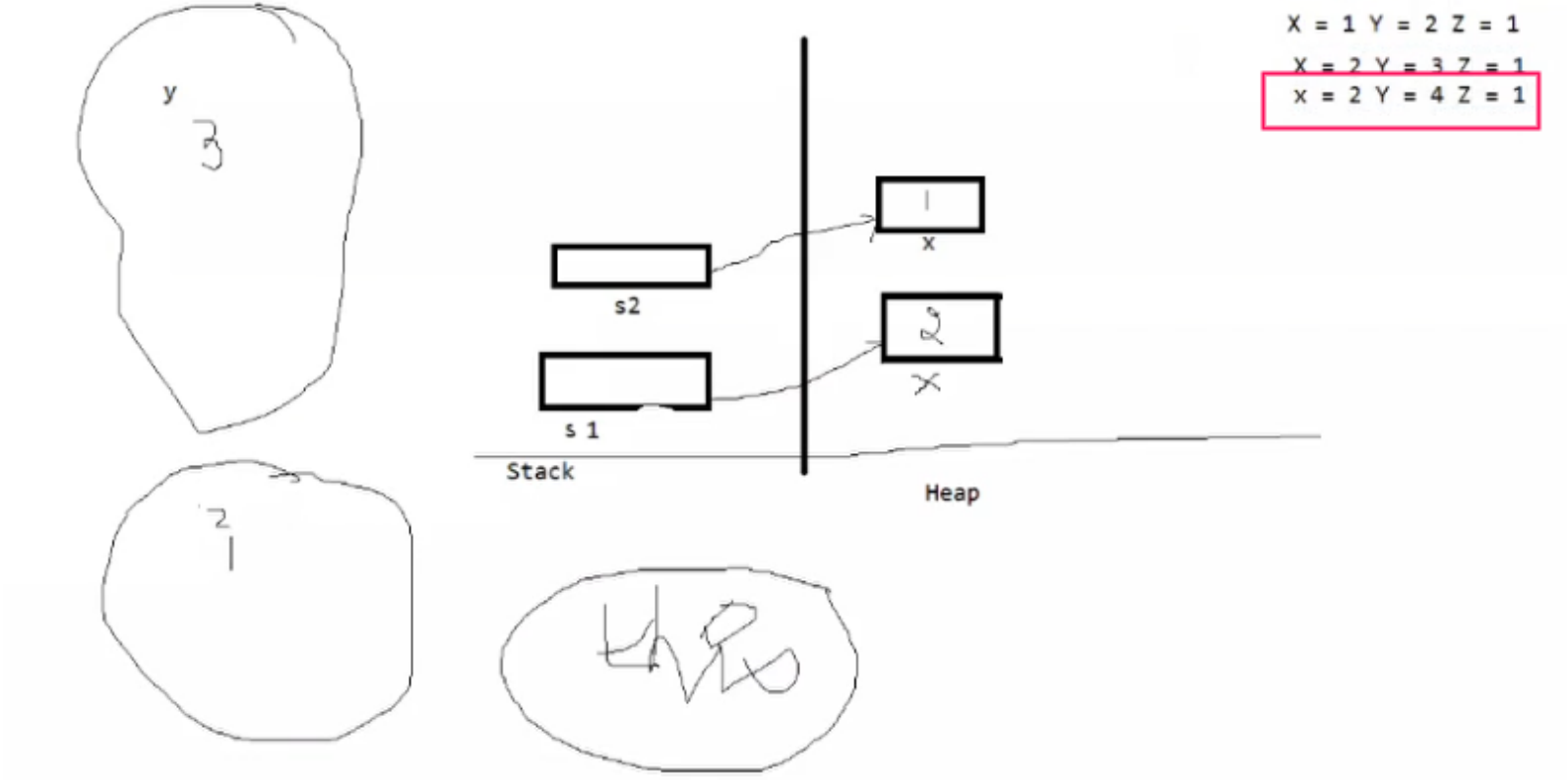
Now s1.show()

x becomes 2, y becomes 3, z becomes 1



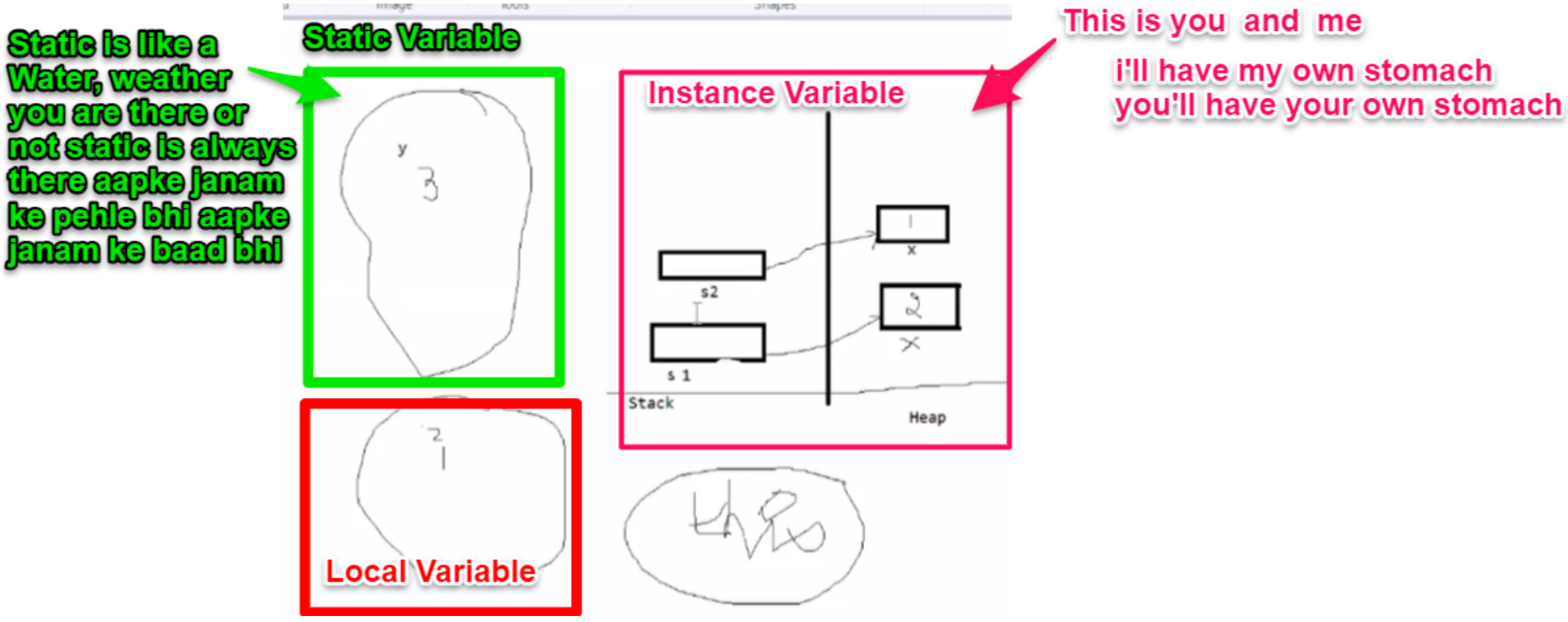
now when i say S2.Show()

X = 1 Y = 1 Z = 1
X = 1 Y = 2 Z = 1
X = 2 Y = 3 Z = 1
X = 2 Y = 4 Z = 1

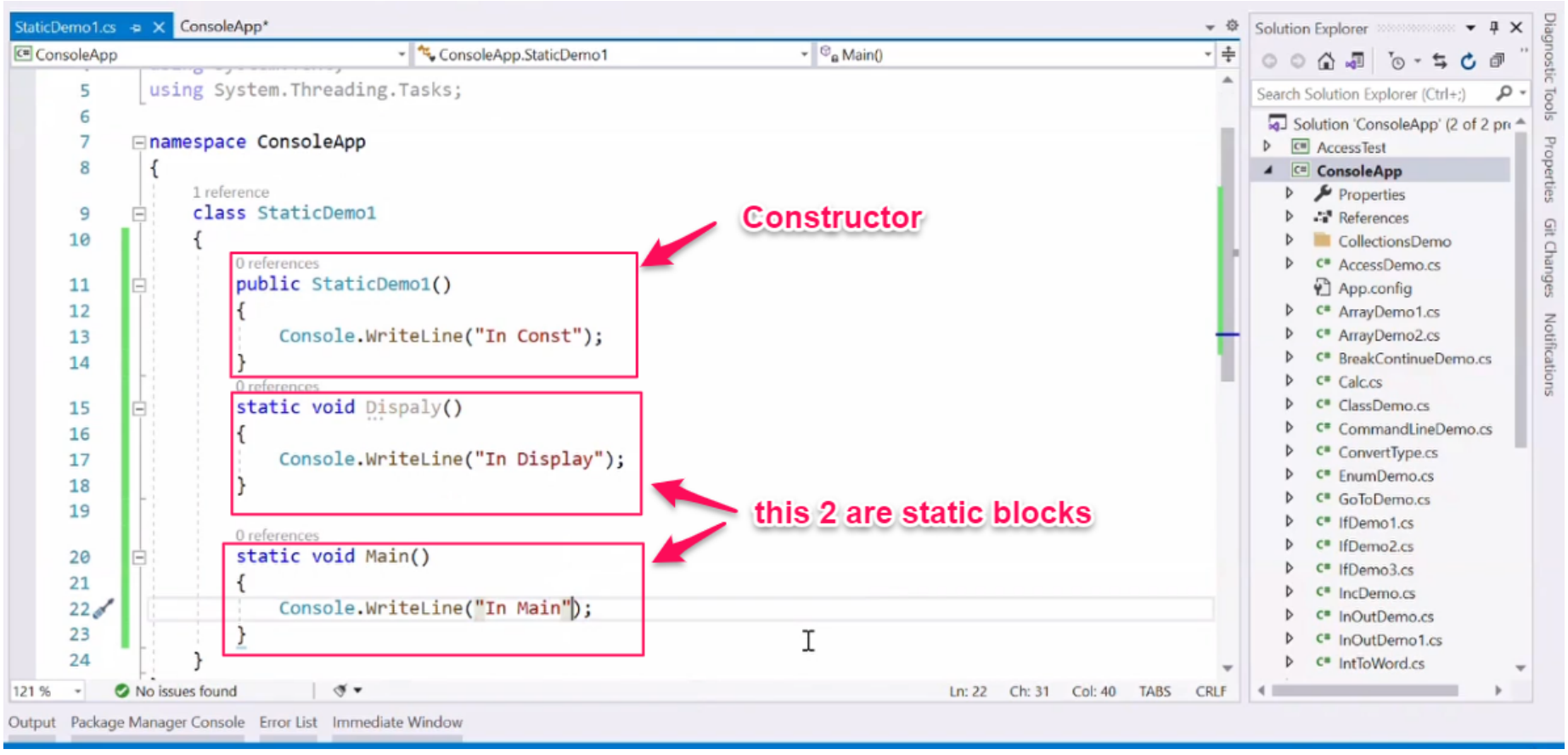


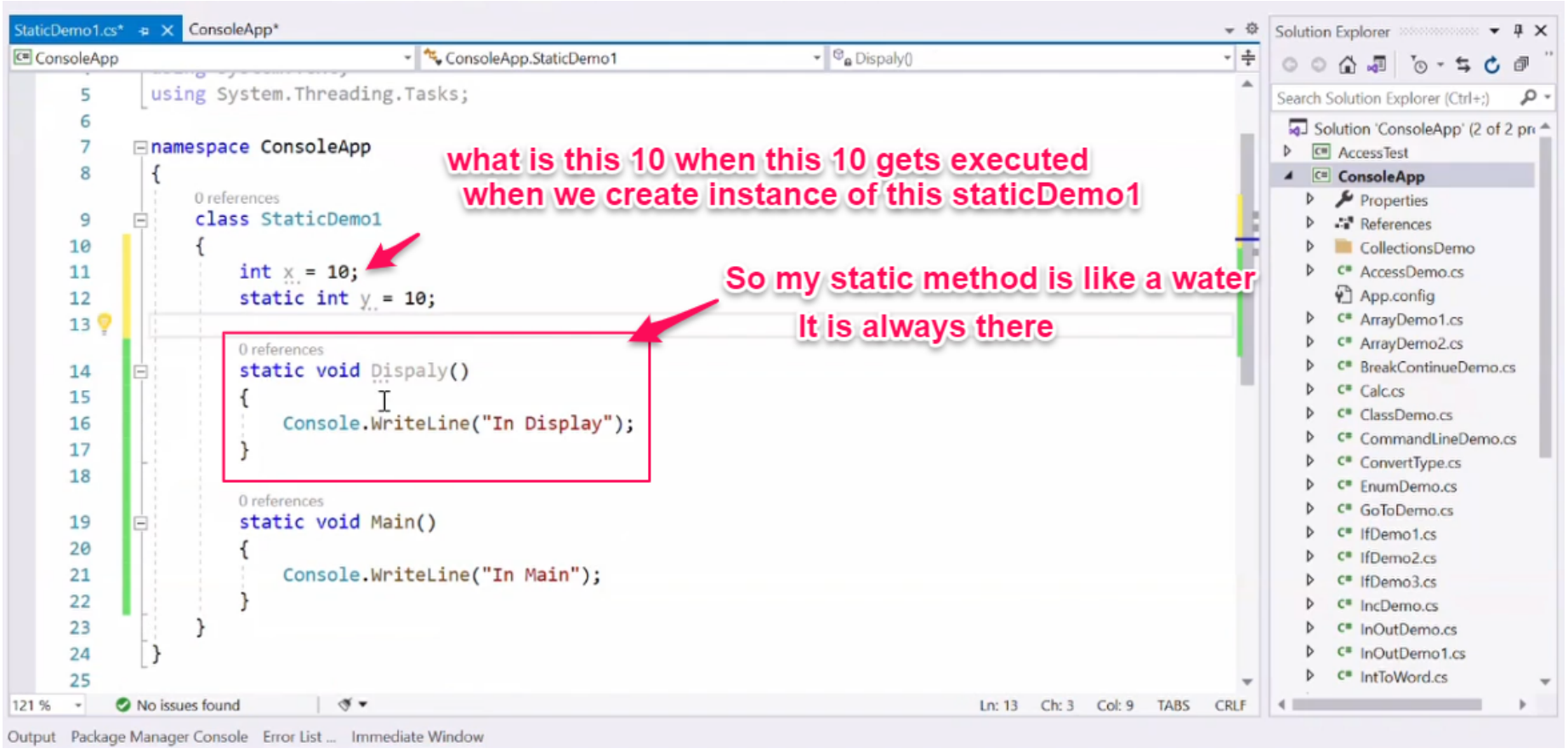
C:\WINDOWS\system32\cmd.exe

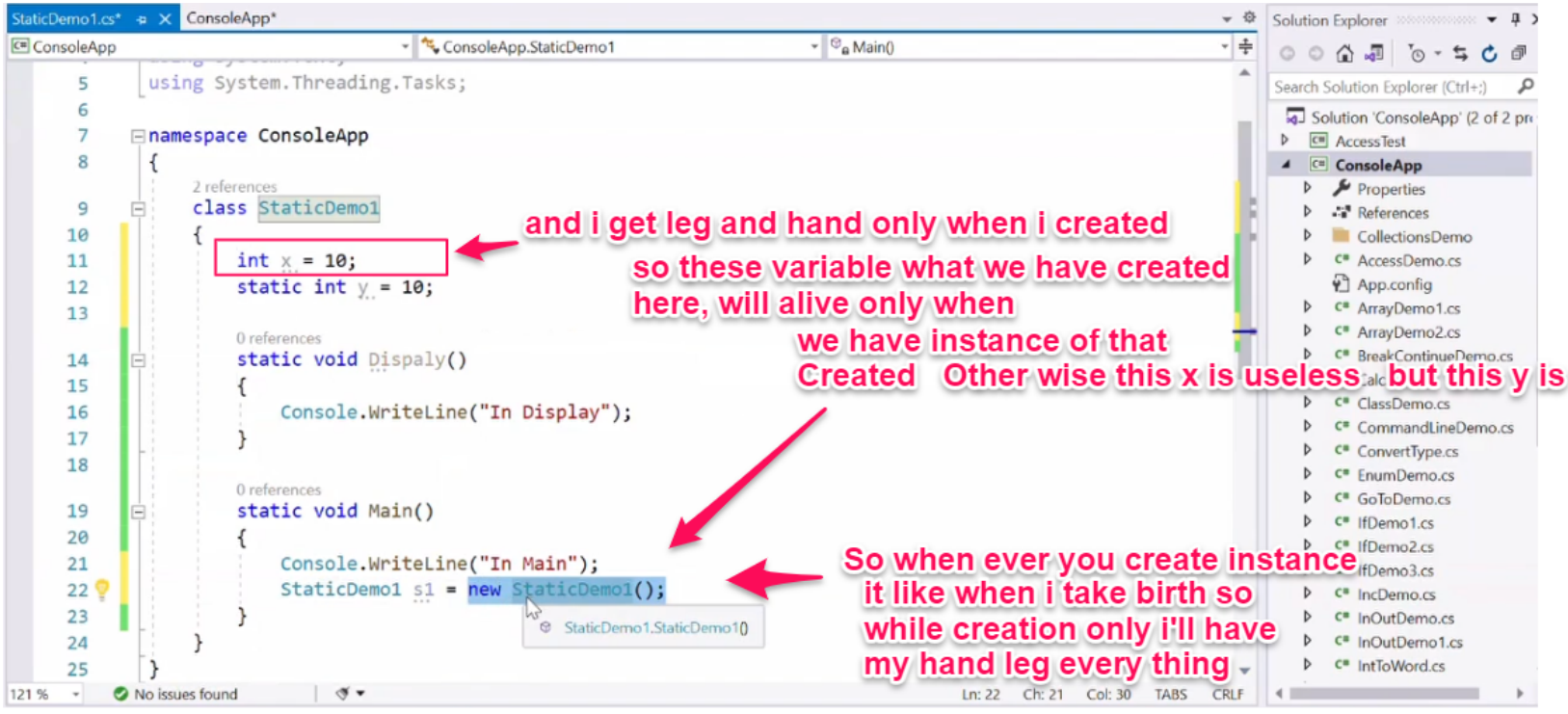
```
X = 1    Y = 1   Z = 1
X = 1    Y = 2   Z = 1
X = 2    Y = 3   Z = 1
X = 2    Y = 4   Z = 1
Press any key to continue . . .
```



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


StaticDemo1.cs
ConsoleApp*
ConsoleApp
ConsoleApp.StaticDemo1
Main()
using System.Threading.Tasks;
namespace ConsoleApp
{
1 reference
class StaticDemo1
{
0 references
public StaticDemo1()
{
Console.WriteLine("In Const");
}
0 references
static void Dispaly()
{
Console.WriteLine("In Display");
}
0 references
static void Main()
{
Console.WriteLine("In Main");
}
}
Constructor
this 2 are static blocks
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApp' (2 of 2 pr
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
121 %
No issues found
Ln: 22
Ch: 31
Col: 40
TABS
CRLF
Output
Package Manager Console
Error List
Immediate Window
Diagnostic Tools
Properties
Git Changes
Notifications


StaticDemo1.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.StaticDemo1
Dispaly()
using System.Threading.Tasks;
namespace ConsoleApp
{
0 references
class StaticDemo1
{
int x = 10;
static int y = 10;
0 references
static void Dispaly()
{
Console.WriteLine("In Display");
}
0 references
static void Main()
{
Console.WriteLine("In Main");
}
}
}
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
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApp' (2 of 2 pr
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
121 %
No issues found
Ln: 13
Ch: 3
Col: 9
TABS
CRLF
Output
Package Manager Console
Error List ...
Immediate Window

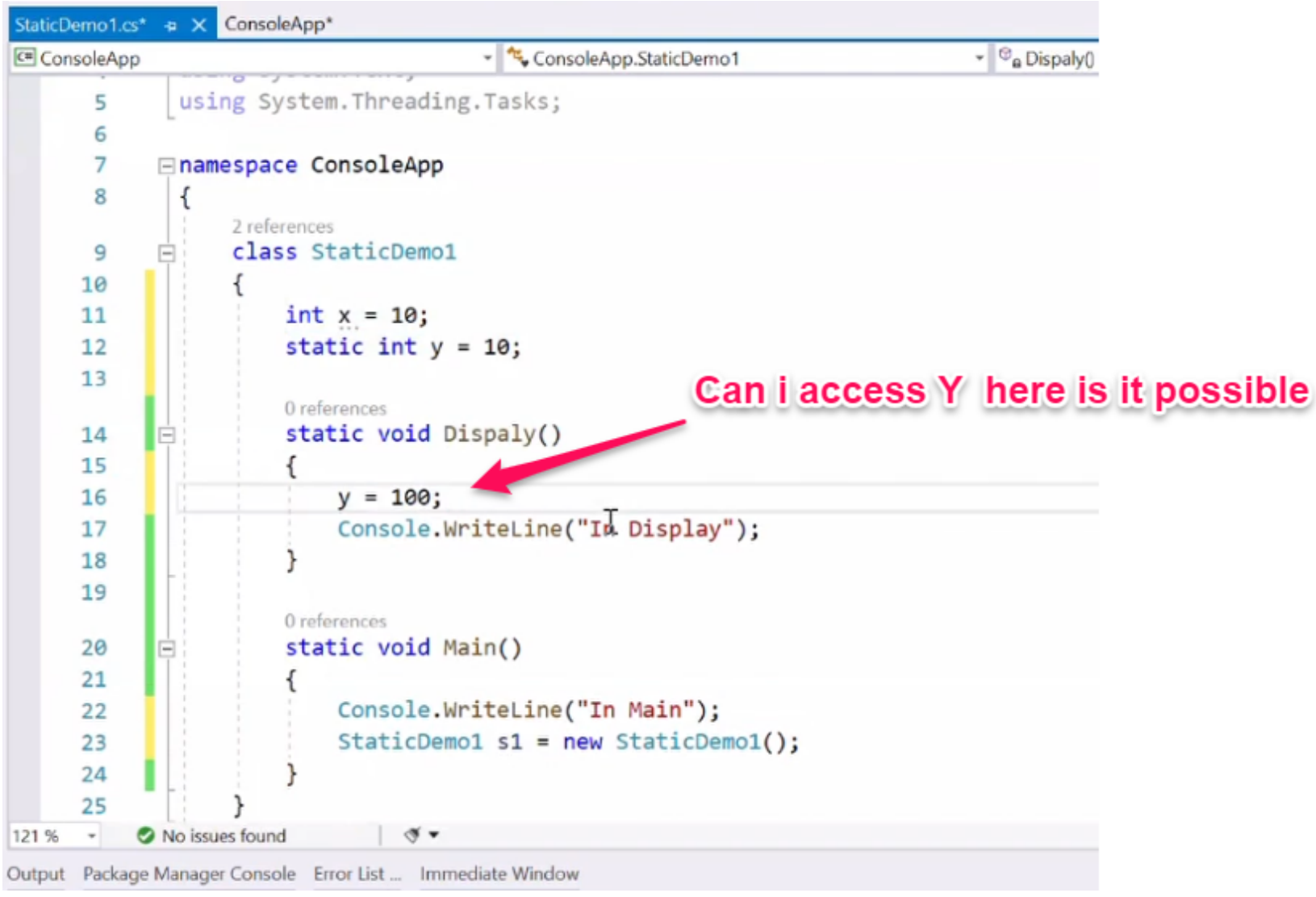


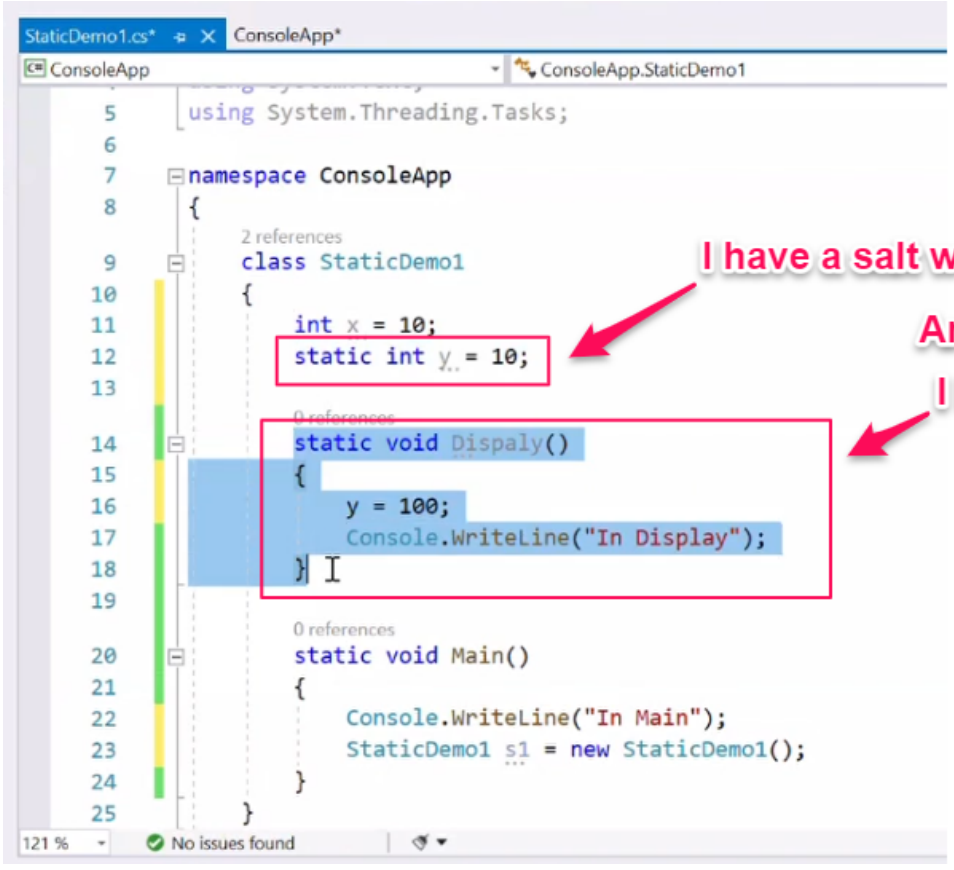
and i get leg and hand only when i created

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*
ConsoleApp*
ConsoleApp
ConsoleApp.StaticDemo1
Dispaly()
using System.Threading.Tasks;
namespace ConsoleApp
{
2 references
class StaticDemo1
{
int x = 10;
static int y = 10;
0 references
static void Dispaly()
{
y = 100;
Console.WriteLine("In Display");
}
0 references
static void Main()
{
Console.WriteLine("In Main");
StaticDemo1 s1 = new StaticDemo1();
}
}
Can i access Y here is it possible
121 %
No issues found
Output
Package Manager Console
Error List ...
Immediate Window



I have a salt water

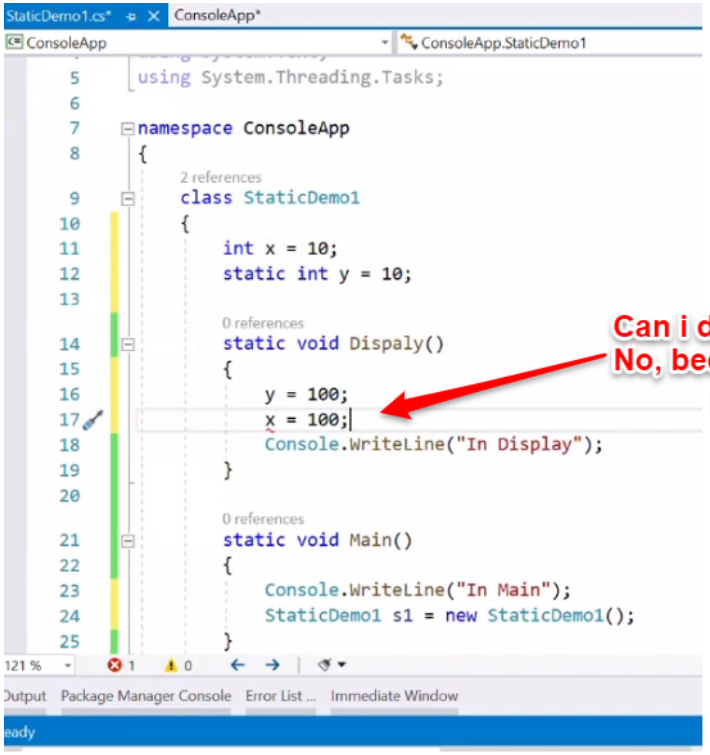
And

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

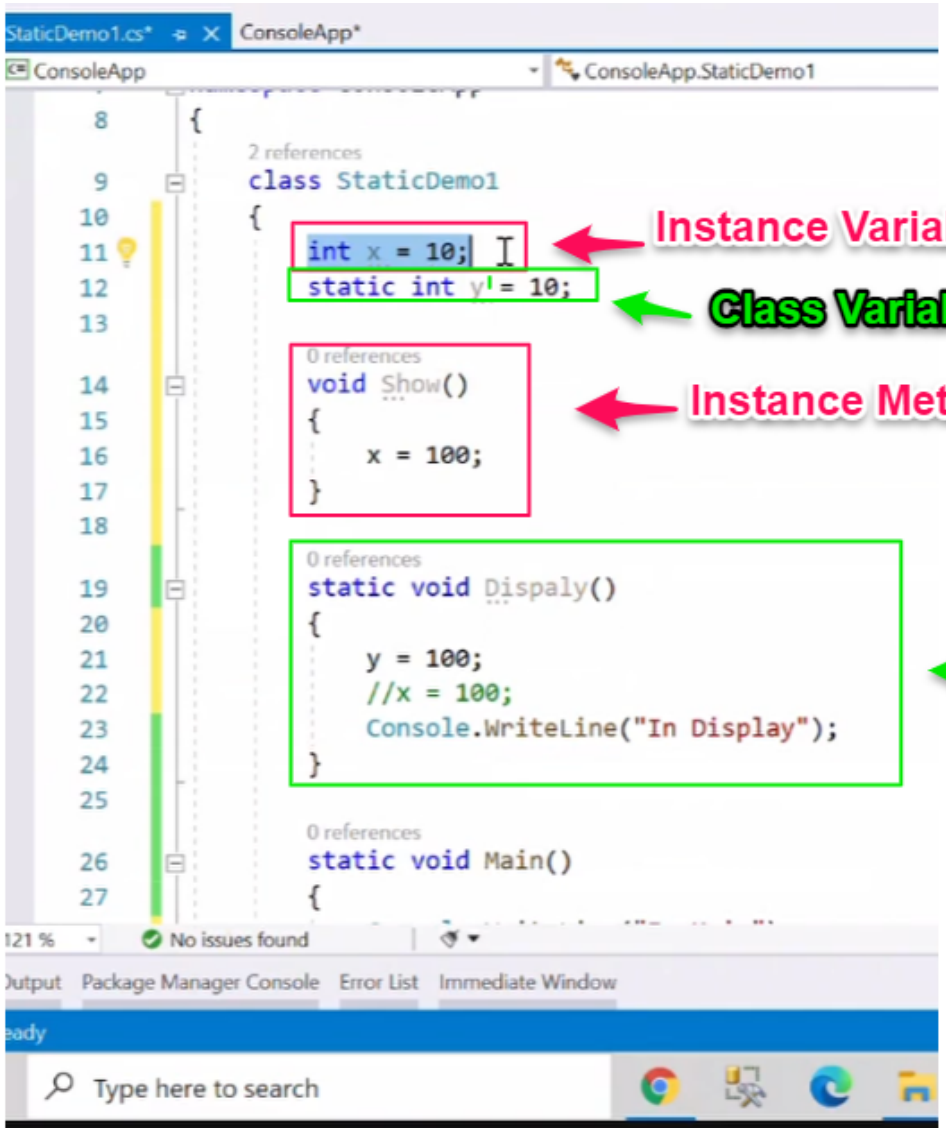


Can i do this ?
No, because x is a instance variable
and x will get life only when you create instance
but water is always there

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 cannpt be accessed
inside the static methods



Instance Variable

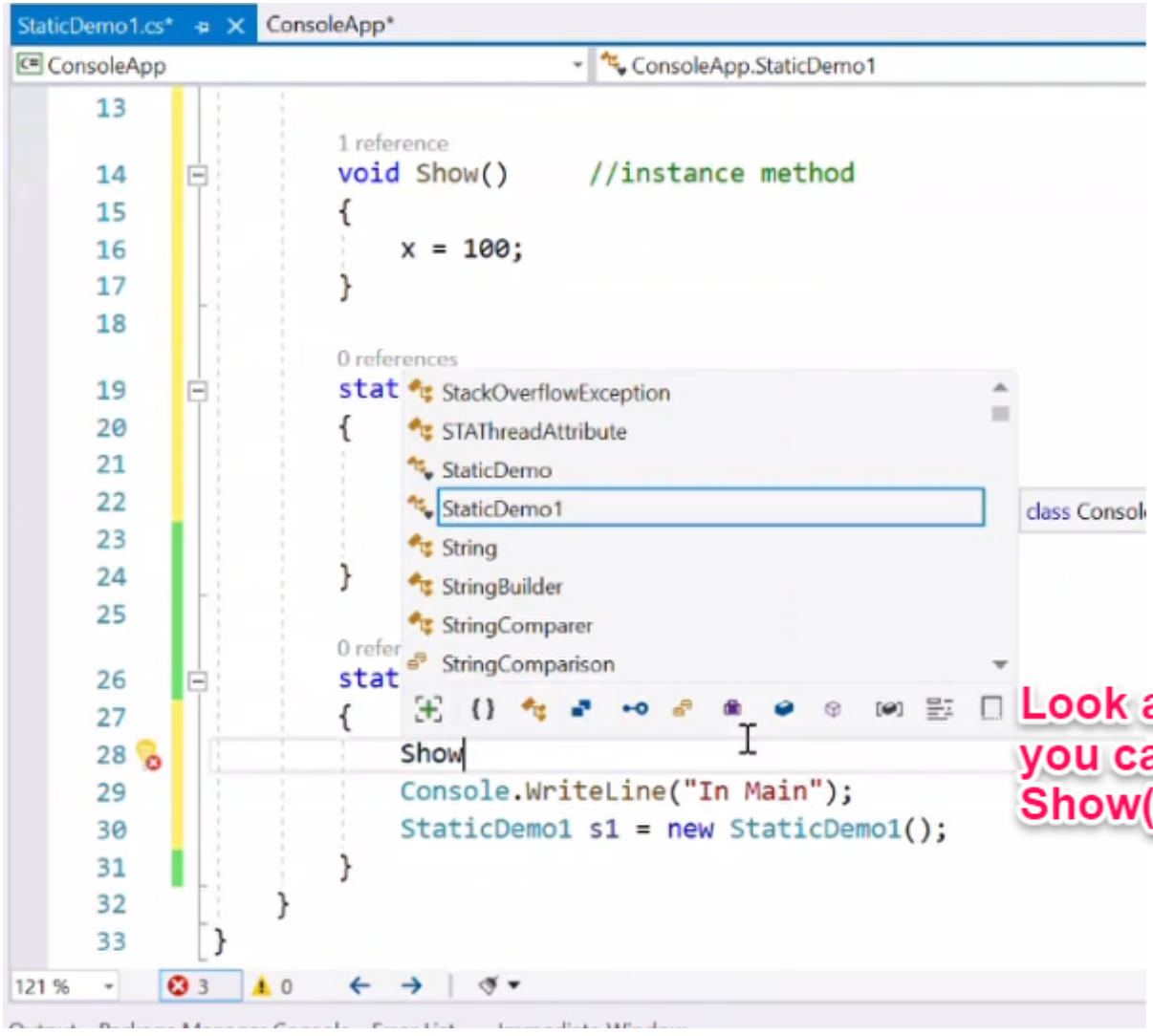
Class Variable

this will be created only when you create object

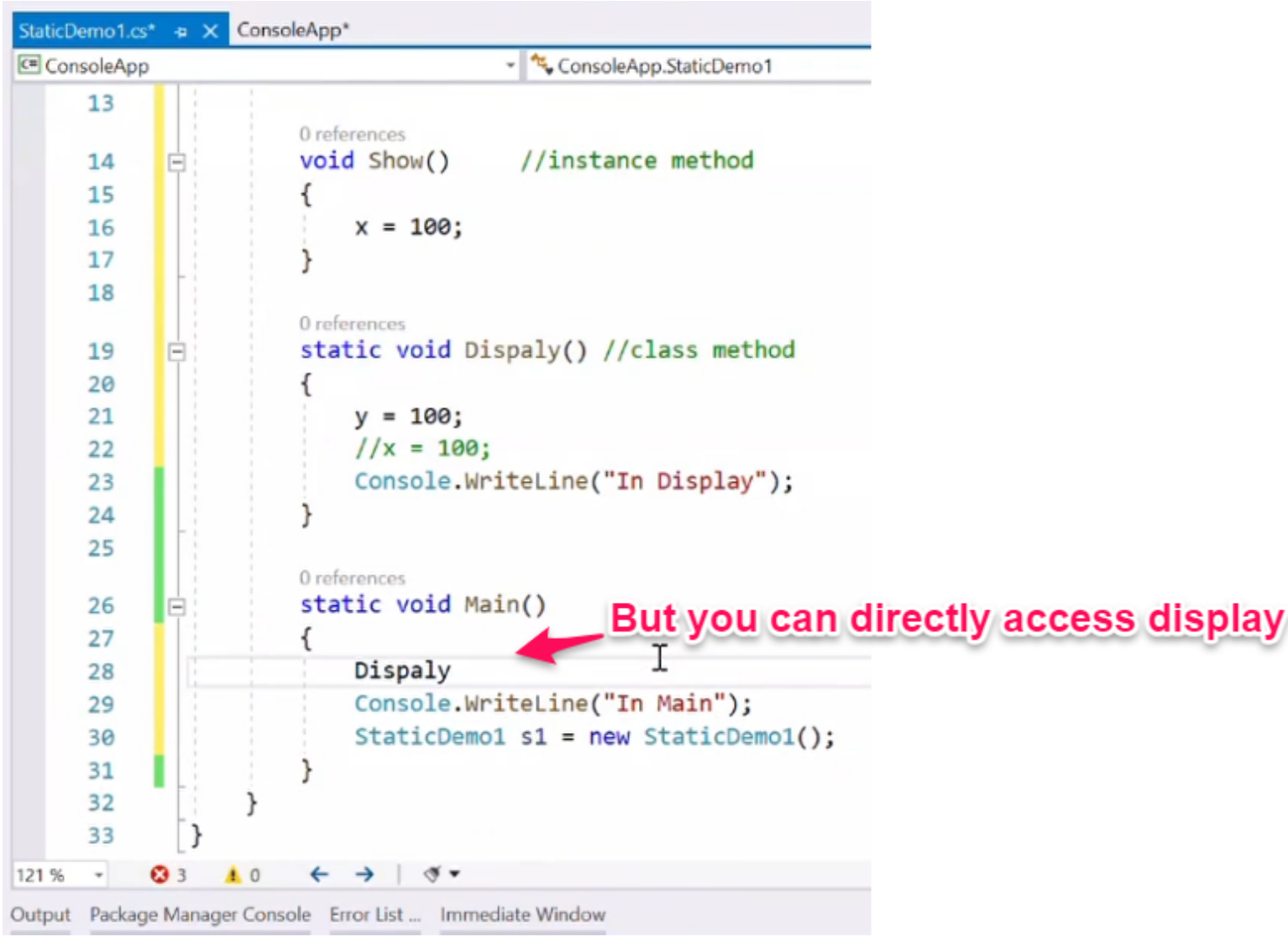
Instance Method

Class Method

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

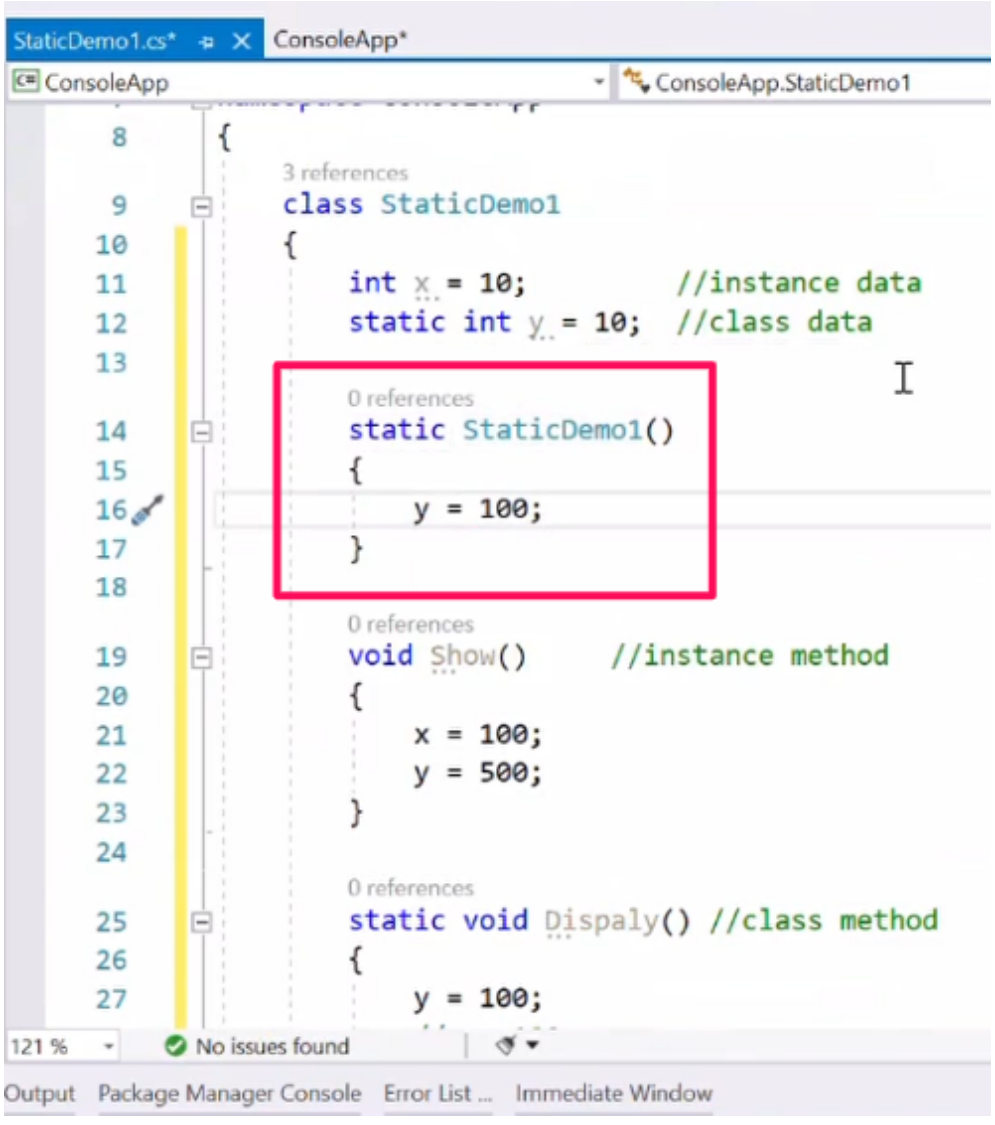


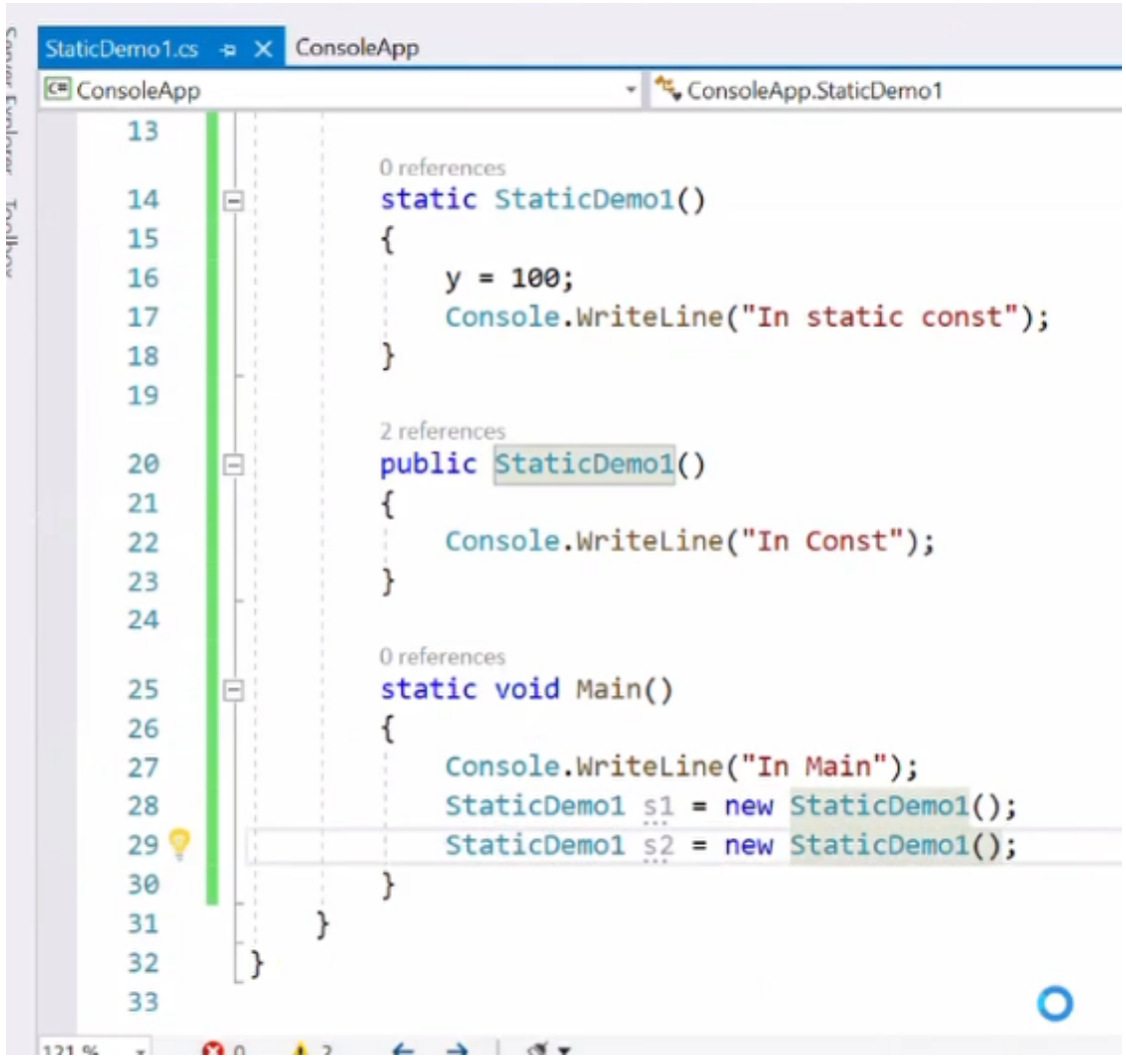
Look at this
you cannot directly say
Show();



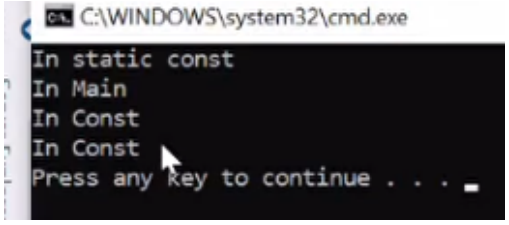
So i have a static member here and i want to initialize them how can I Initialize ?

so just create a static ctor (constructor)





What will be the output ?



you can see that i'm creating multiple instances
and even thoe 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 weather 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

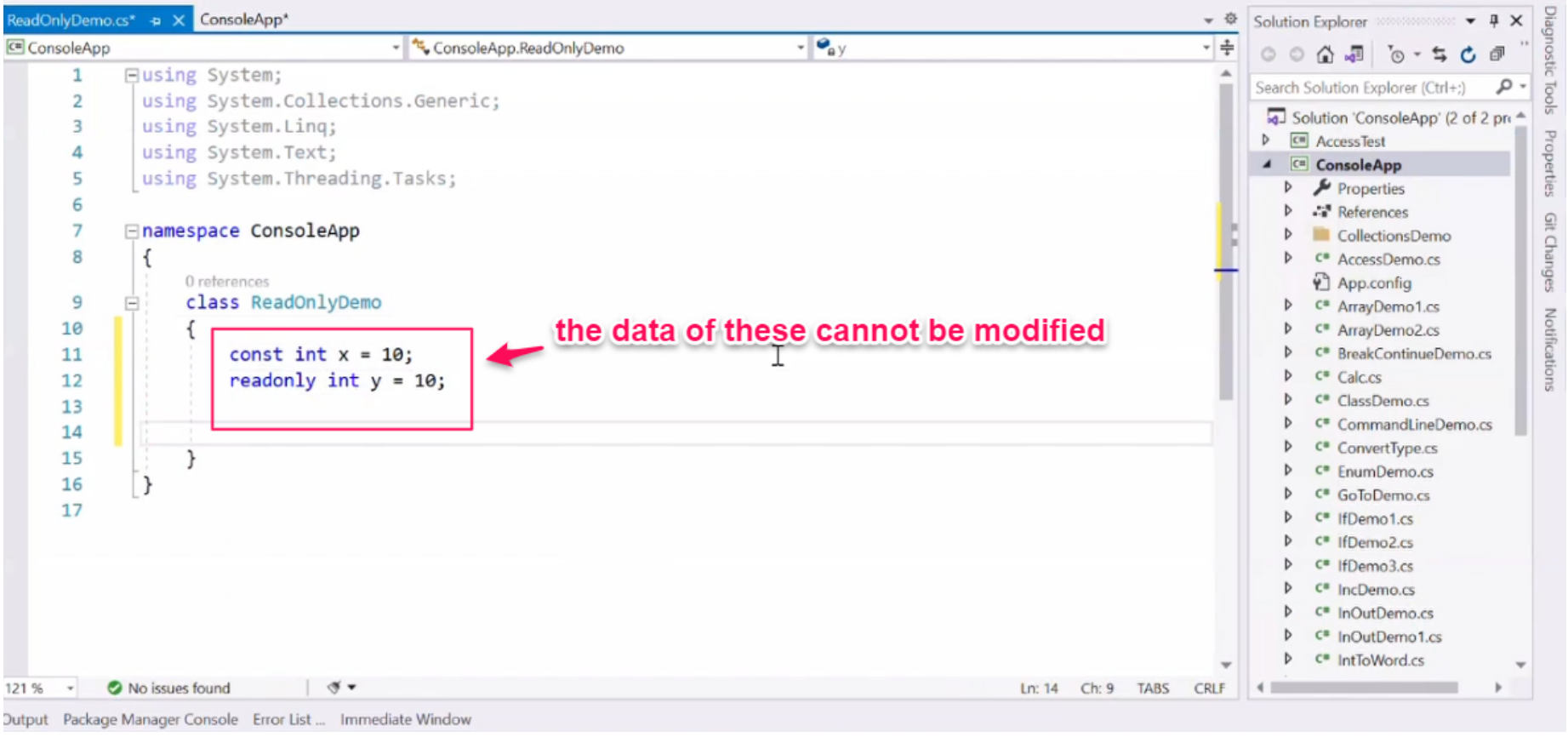
Declaring Readonly Variables and Constants

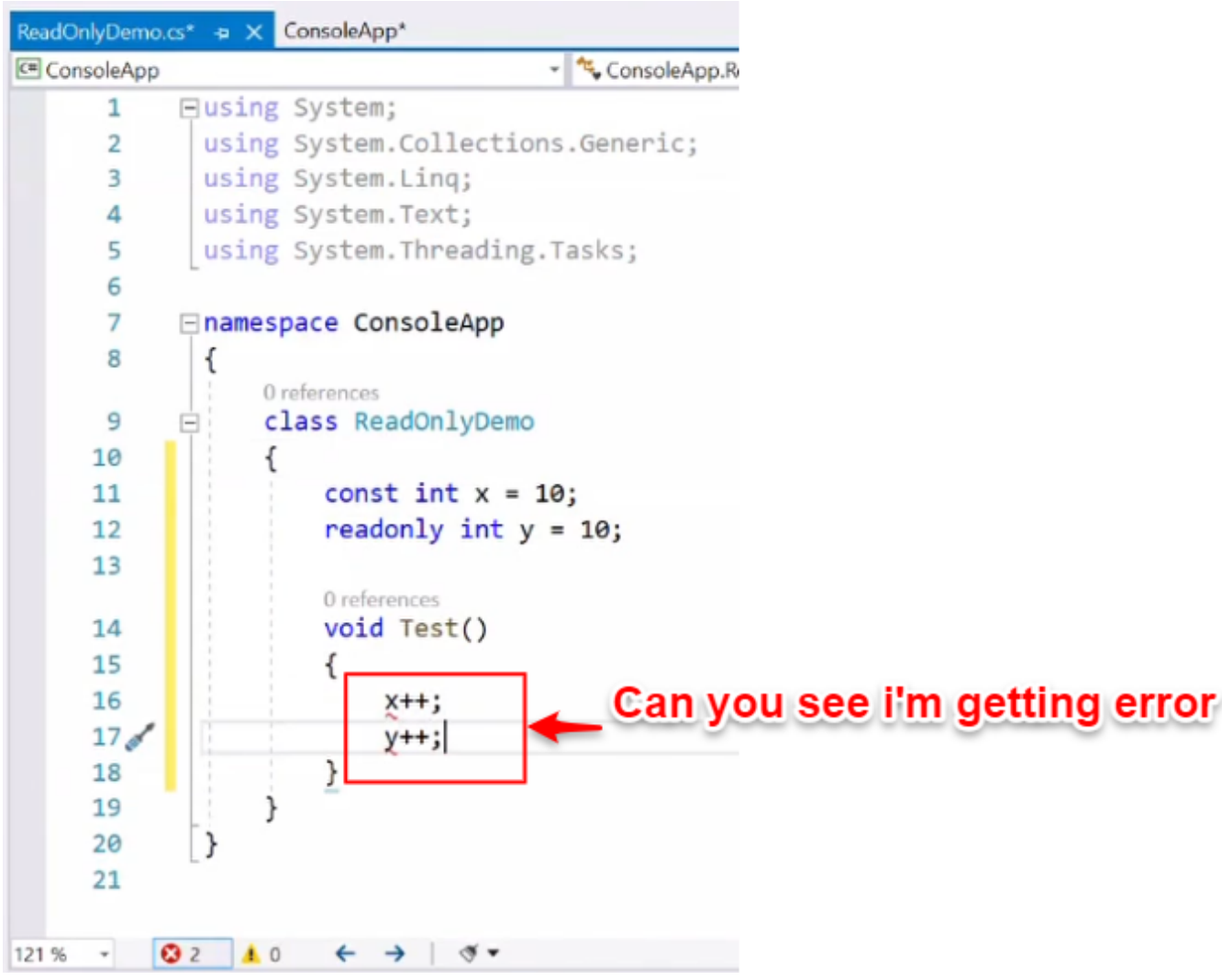
Constant Variable

- 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

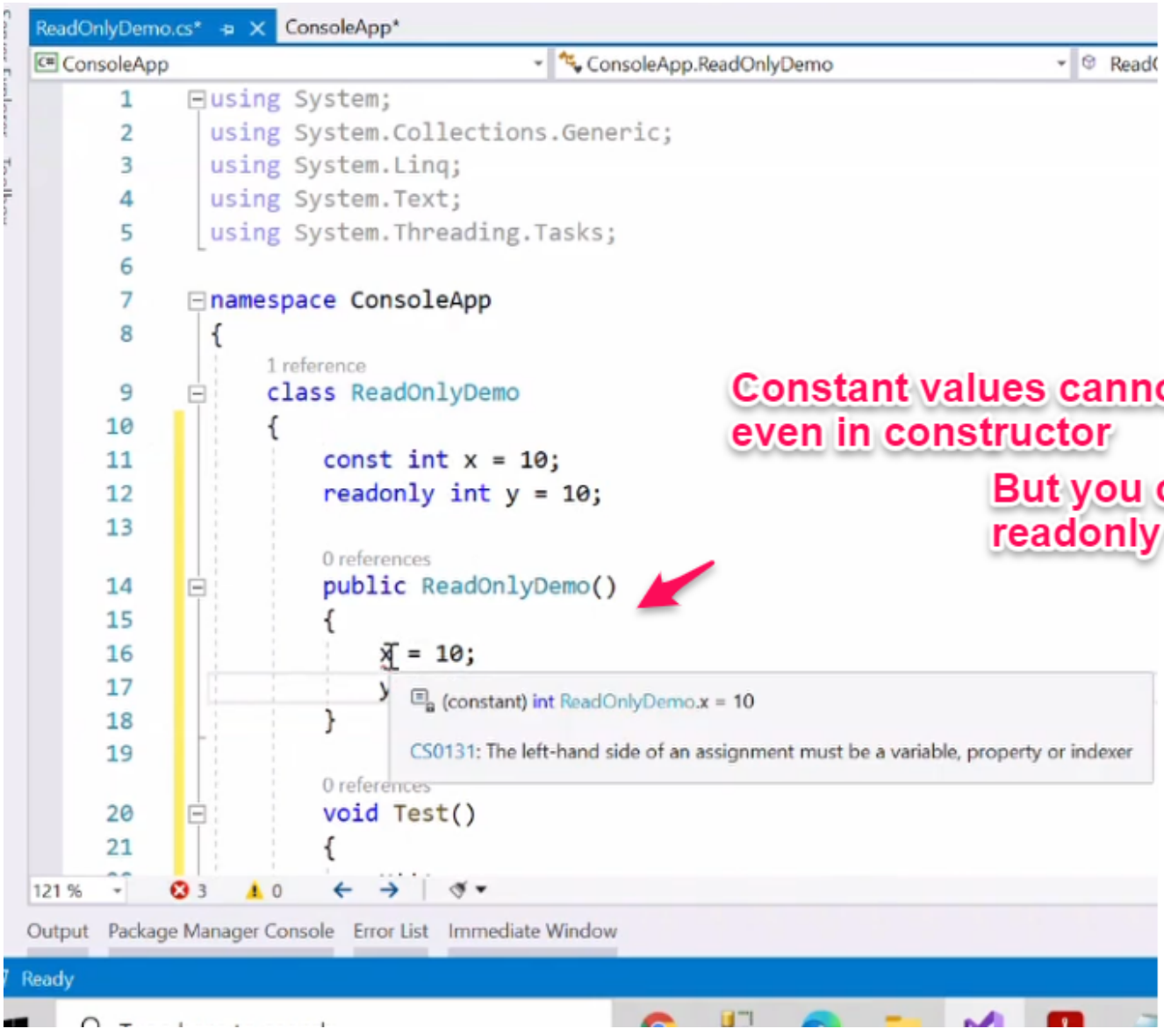
Read-only variable


ReadOnlyDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.ReadOnlyDemo
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
0 references
class ReadOnlyDemo
{
const int x = 10;
readonly int y = 10;
}
}
the data of these cannot be modified
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApp' (2 of 2 pr
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
Diagnostic Tools
Properties
Git Changes
Notifications
121 %
No issues found
Ln: 14
Ch: 9
TABS
CRLF
Output
Package Manager Console
Error List ...
Immediate Window



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

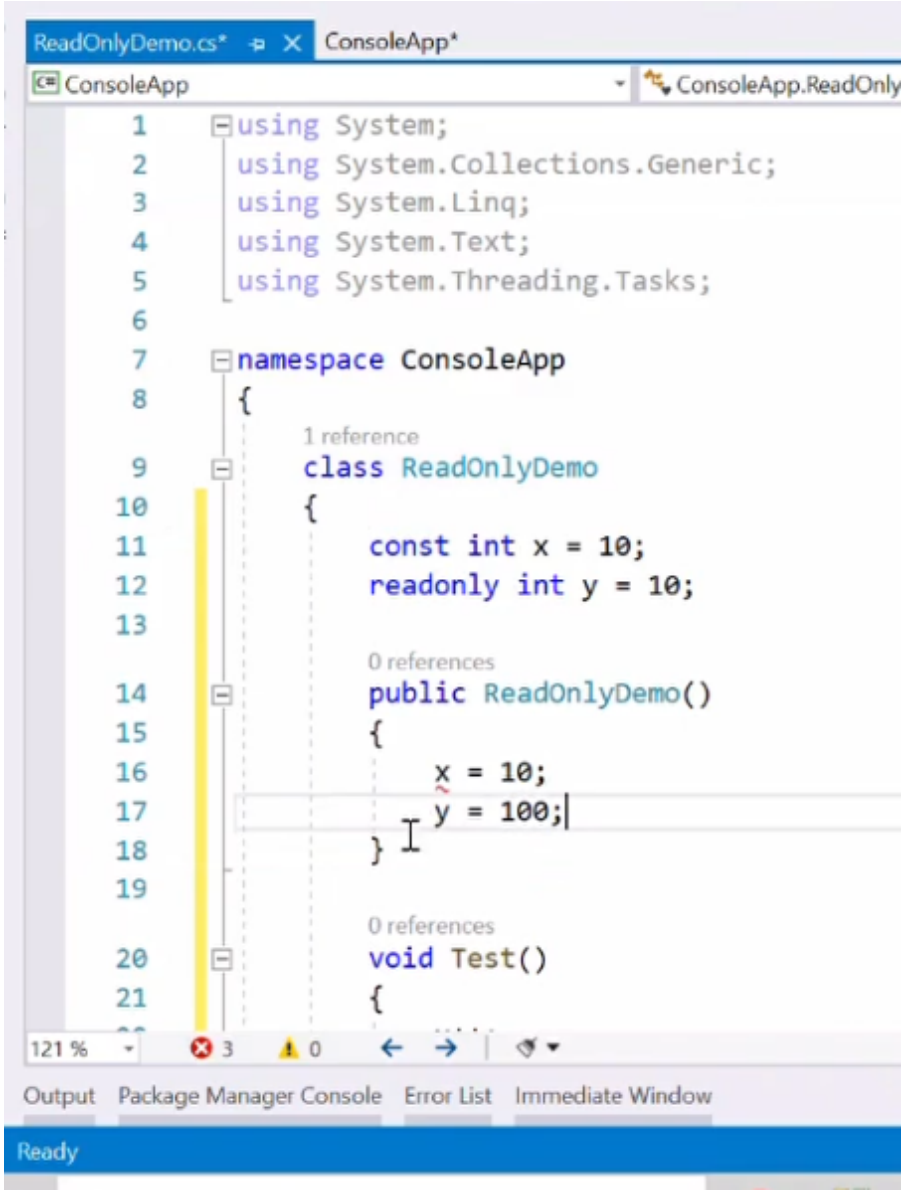
because the word itself says vary meaning we can change the value
and in constants we cannot modify or change the value
We cannot modify the value of Constant and Readonly



Constant values cannot be modified even in constructor

But you can initialize the readonly inside the constructor

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


ReadOnlyDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.ReadOnly
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
1 reference
class ReadOnlyDemo
{
const int x = 10;
readonly int y = 10;
0 references
public ReadOnlyDemo()
{
x = 10;
y = 100;
}
0 references
void Test()
{
121 %
3
0
Output
Package Manager Console
Error List
Immediate Window
Ready

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*
ConsoleApp*
ConsoleApp
ConsoleApp.ReadC
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
1 reference
class ReadOnlyDemo
{
const int x;
readonly int y;
0 references
public ReadOnlyDemo()
{
y = 100;
}
0 references
void Test()
{
x++;
121 %
2
0
Output
Package Manager Console
Error List ...
Immediate Window
eady
Type here to search