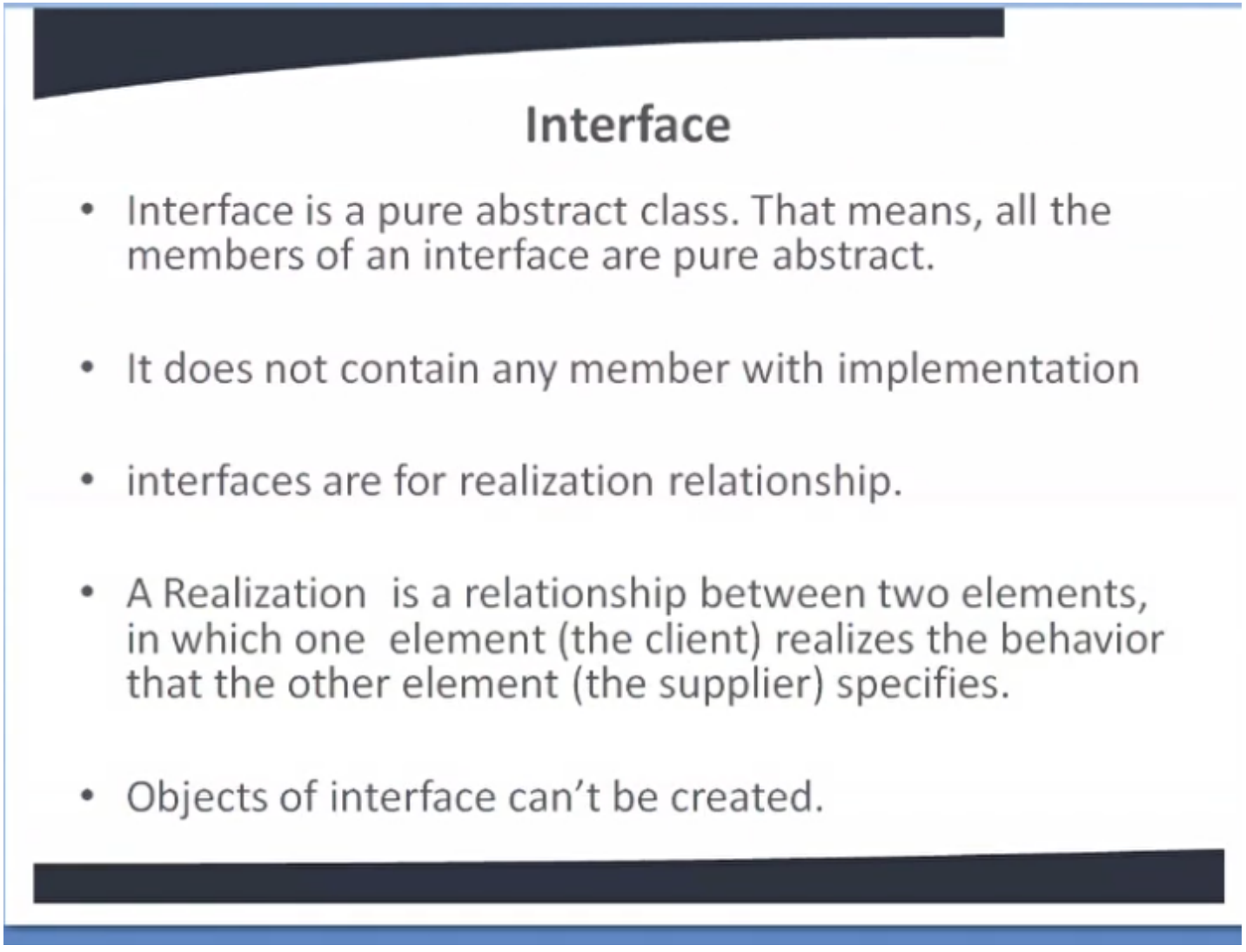
C# - 11

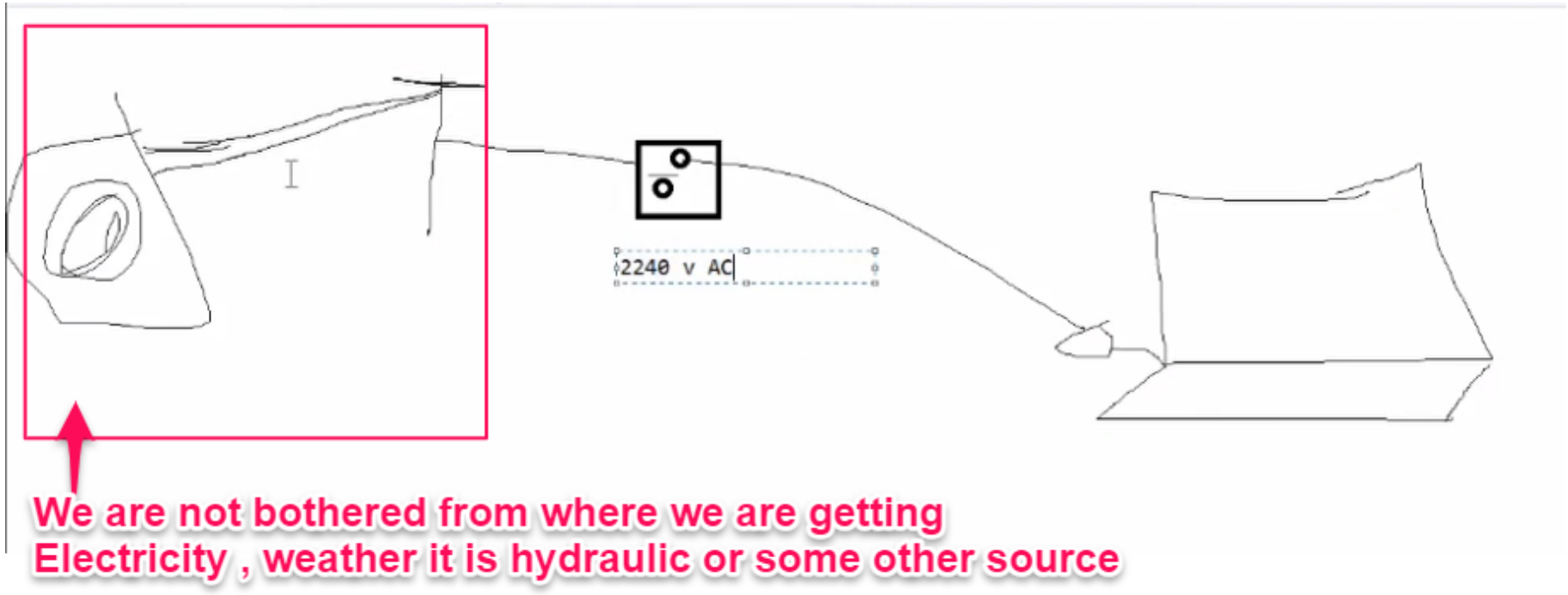
Interface



In General Interface is to Identify the behaviours

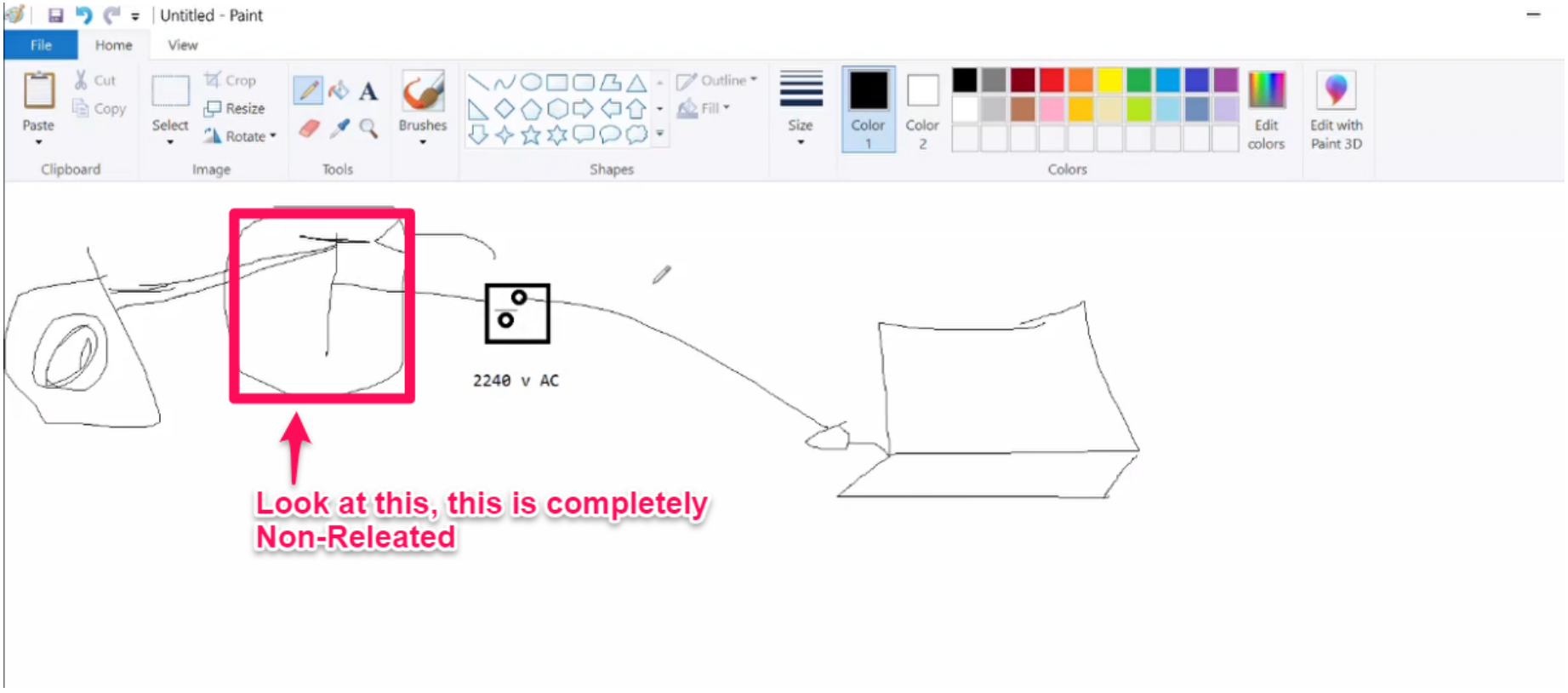
Suppose I have a TV

When you Count TV as a TV -------When it does the job of a TV , when it looks like a TV
we have a laptop with us, it has a adaptor which connects to 3 pin socket, if you are connecting to 2 pin pls change it to 3 pin other wise your harddisk will corrupt, make sure it connects to ground

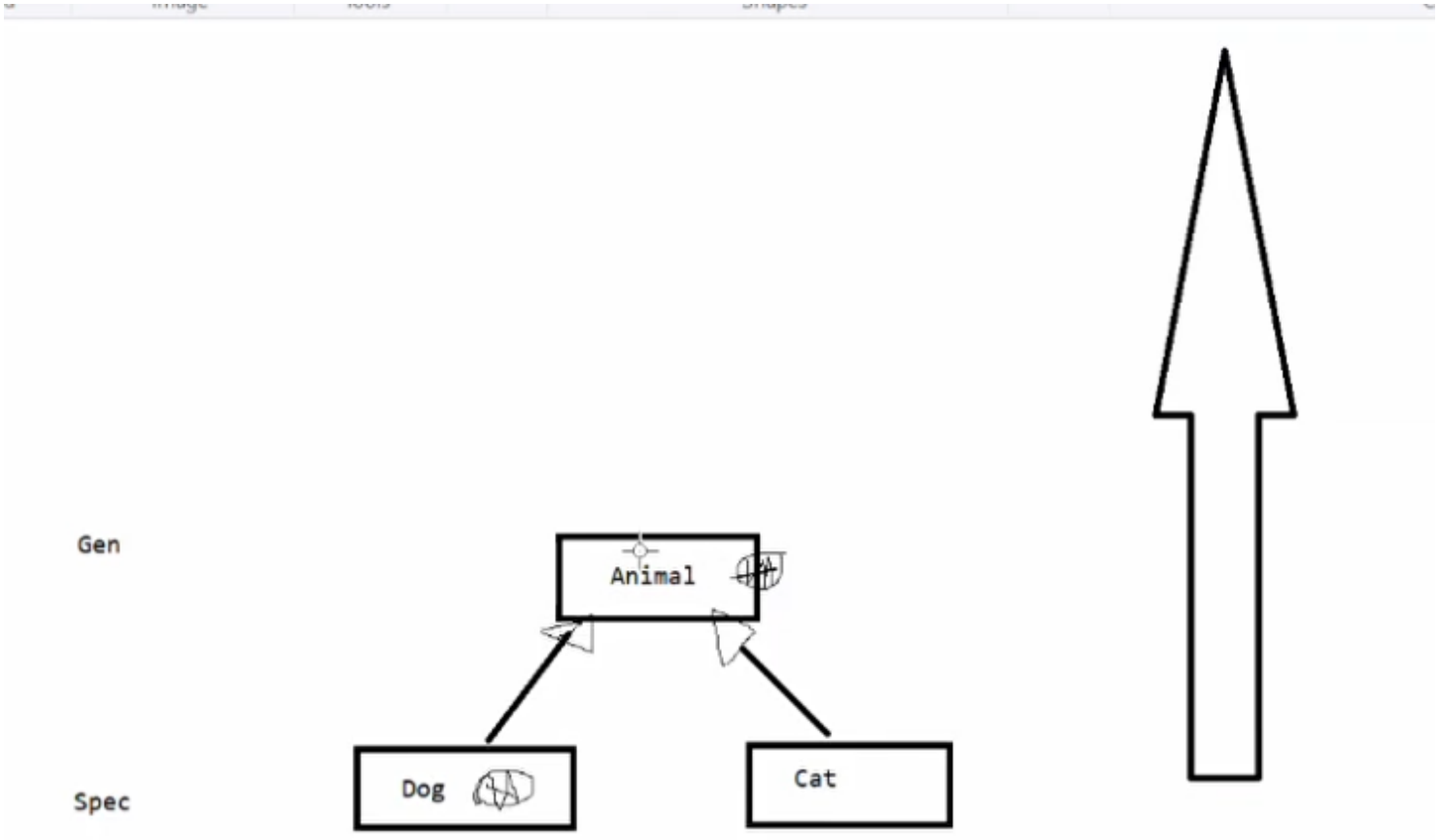


At the same time does this guy who is supplying electricity will he be bothered that what device you are connecting to, weather it is TV or AC or .....

So this 2 pin or 3 Pin socket be work as a Interface to connect to non related and I'm not at all bothered to non related objects



In Interface what we do, It is also a kind of Inheritance
So Usually this Interface comes in Designing
and we have already seen this



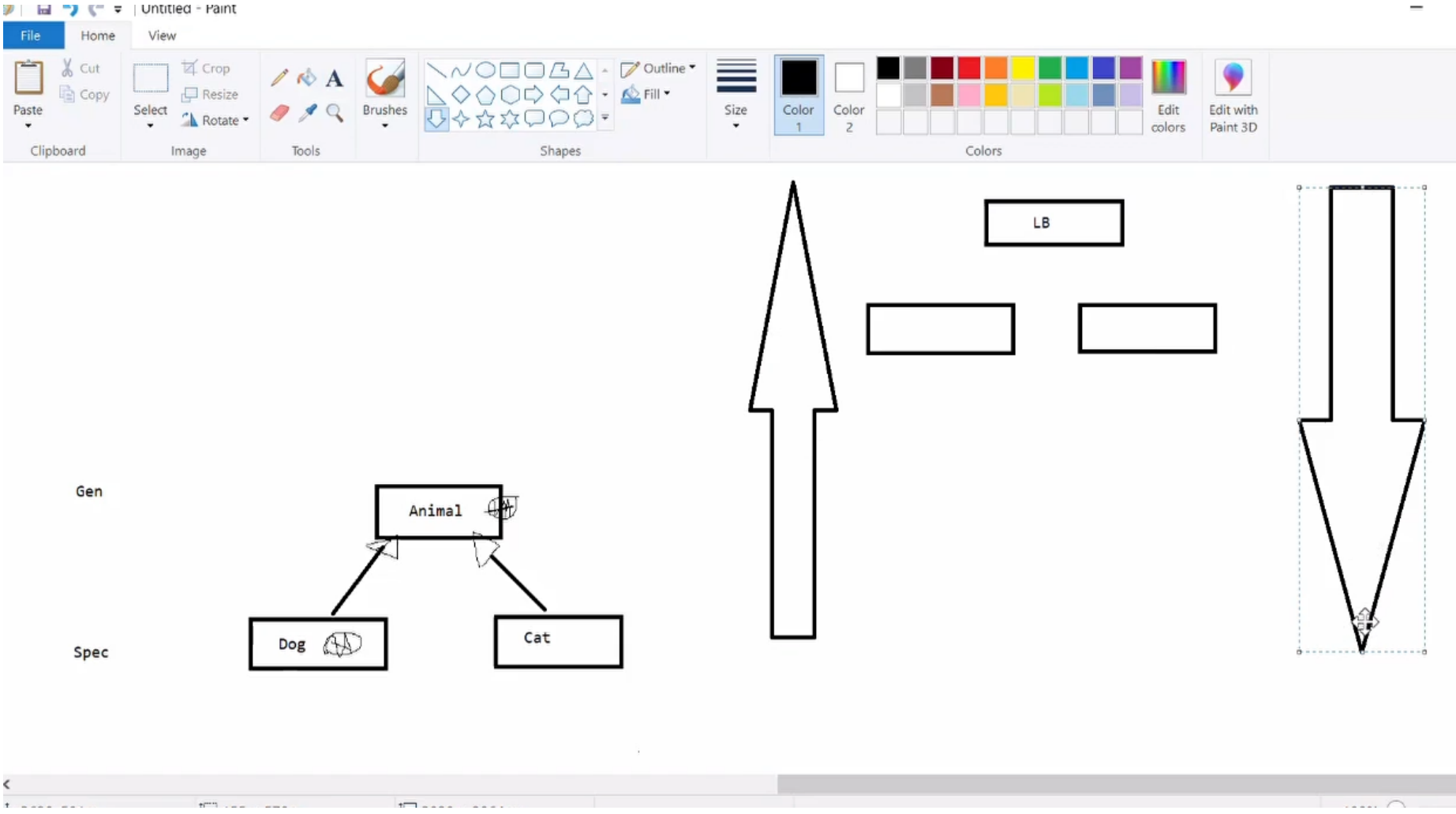
Here we saw that we are identifying dog and cat and when we saw the common functionality we are bringing that in animal
So this is Specialization to generalization this will always be between 2 different classes, which we have already seen

What if I already know the Big Pitcher of the project

Usually what happens if it is a new project, we will be knowing the big pitcher of the project, what exactly project is all about, what are the behaviours of it , I know the complete pet store
Its, not that I only work with the dog and cat
so what I do
1st i'll identify that my pet store will deal with Animal and Birds
Now what is common between animal and birds
-------they both are Living Being.
What is the behaviour of Living beings ?
when it has 3 charters ?
------ **Life Span**, any object will have a lifespan
------ **Reproduction**
So when I say like living being I'll write only the abstract information of it
So 1st We'll identify all the behaviour down the line then we'll start implementing them
as and when we are inheriting them, we are implementing them
meaning 1st we'll go with the generalized behaviour------we'll write a note of all the generalized behaviour then we'll start writing the specialization at the bottom

So In Inheritance it is Specialization to Generalization

But in Interface it is Generalization to Specialization

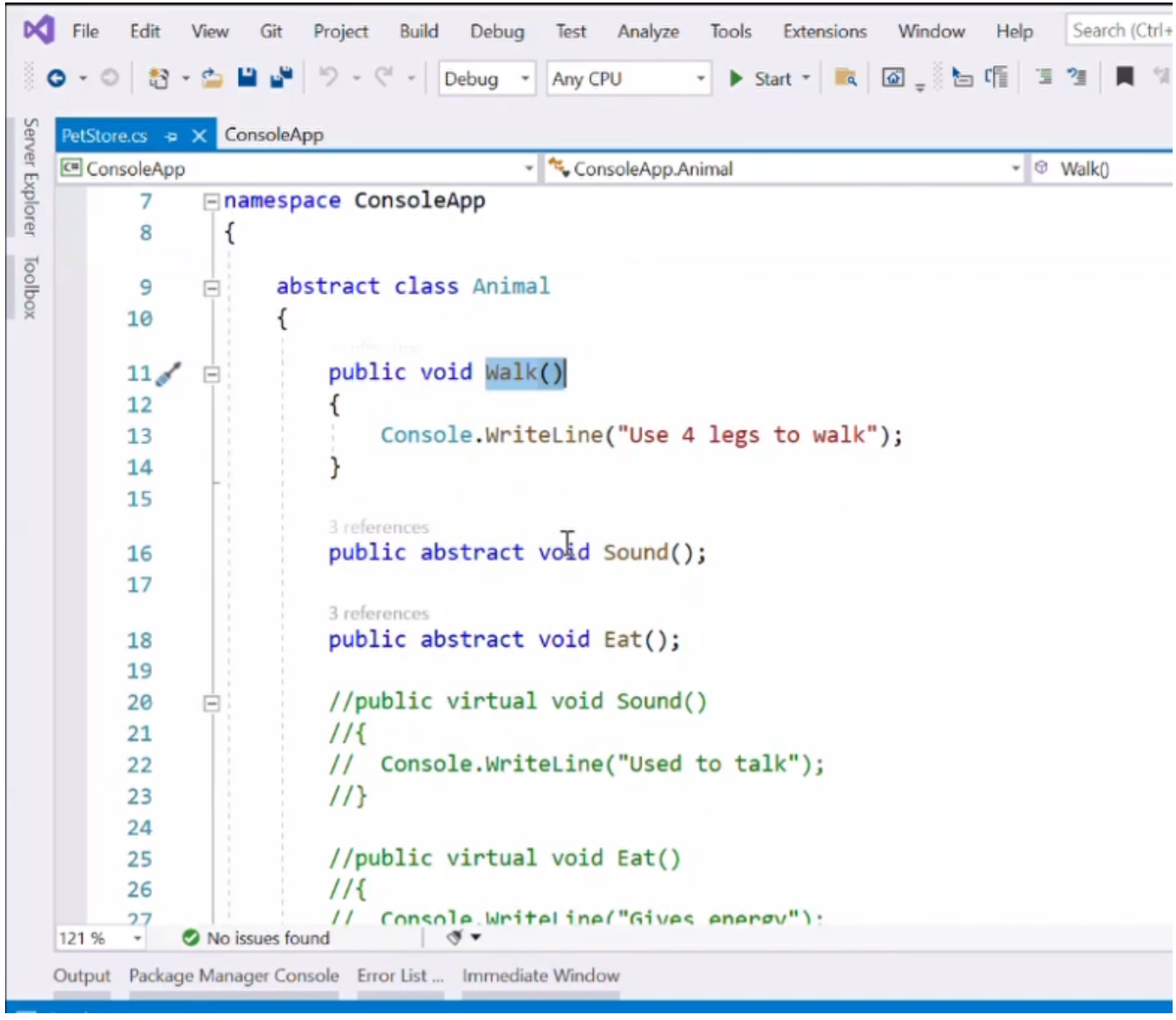


So When ever it is coming from top to bottom it is through Interfaces

in Interface we'll define all the properties then we'll going to Inherit

So we can use both Inheritance and Interface

But always the Interface relationship is better, because in Inheritance, it is always tight coupling
for ex: if i go to pet store
We have already defined walk() of a particular animal


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+
Debug
Any CPU
Start
Server Explorer
Toolbox
PetStore.cs
ConsoleApp
ConsoleApp
ConsoleApp.Animal
Walk()
namespace ConsoleApp
{
abstract class Animal
{
public void Walk()
{
Console.WriteLine("Use 4 legs to walk");
}
3 references
public abstract void Sound();
3 references
public abstract void Eat();
//public virtual void Sound()
//{
// Console.WriteLine("Used to talk");
//}
//public virtual void Eat()
//{
121 %
No issues found
Output
Package Manager Console
Error List ...
Immediate Window

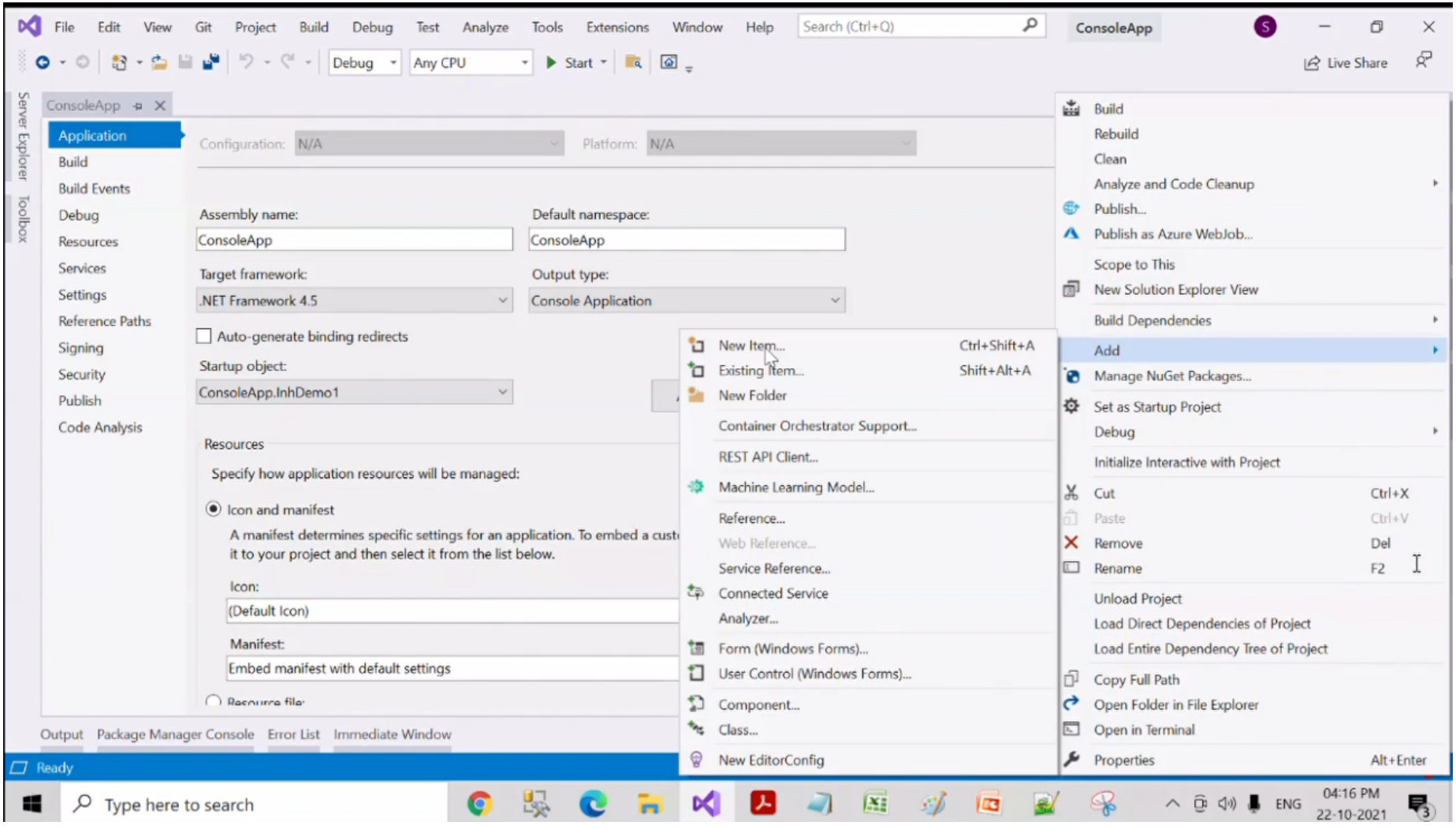
but tomorrow I want to introduce a new method and want to make sure only couple of classes should inherit it, So I cannot do that

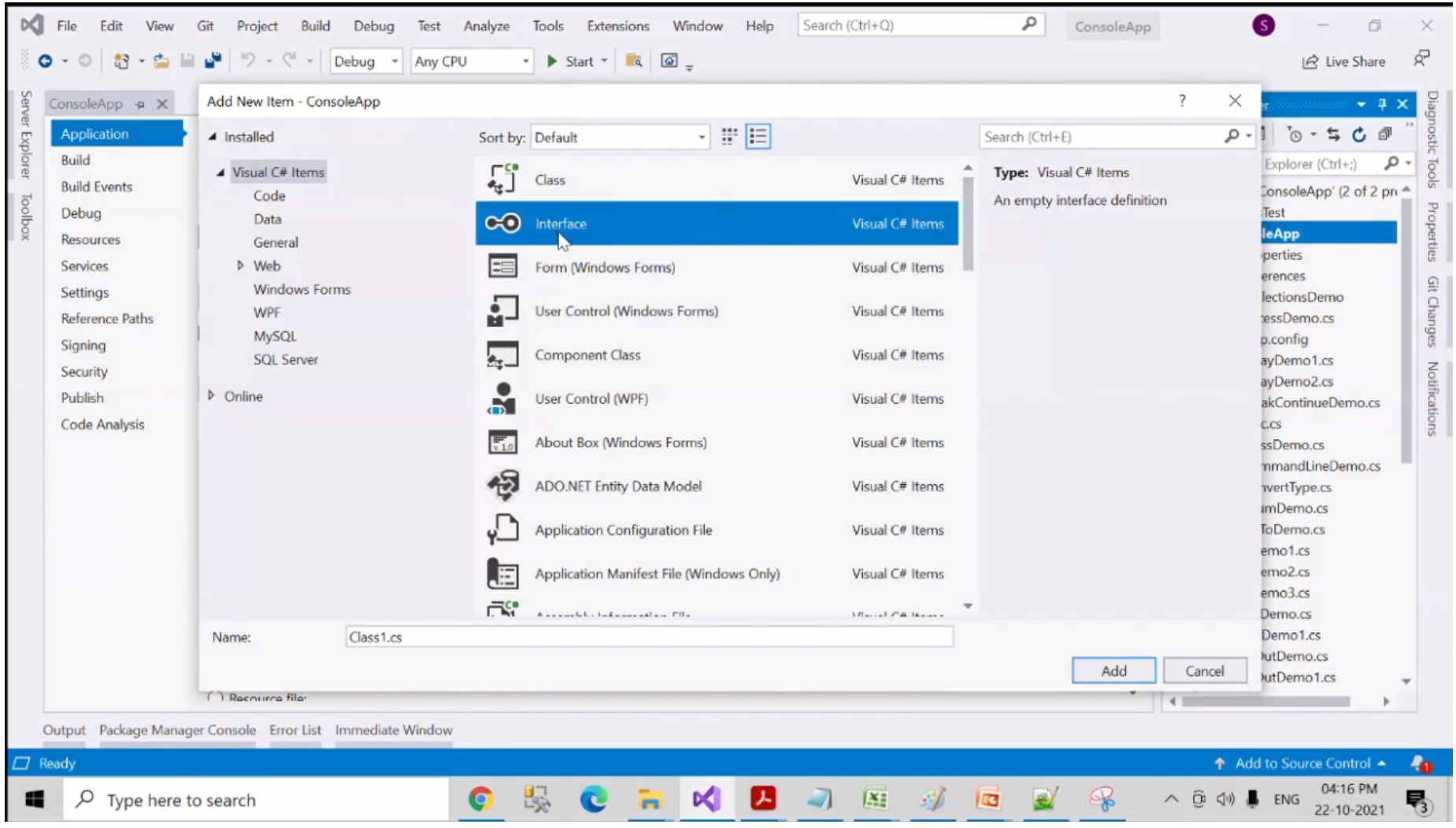
or

I want to modify this walk only for couple of classes , Inherited classes down the line , So i cannot do that
meaning if I add something here or if i subtract something here that affect all the classes which is inheriting it and that becomes a disadvantage because its a tight coupling

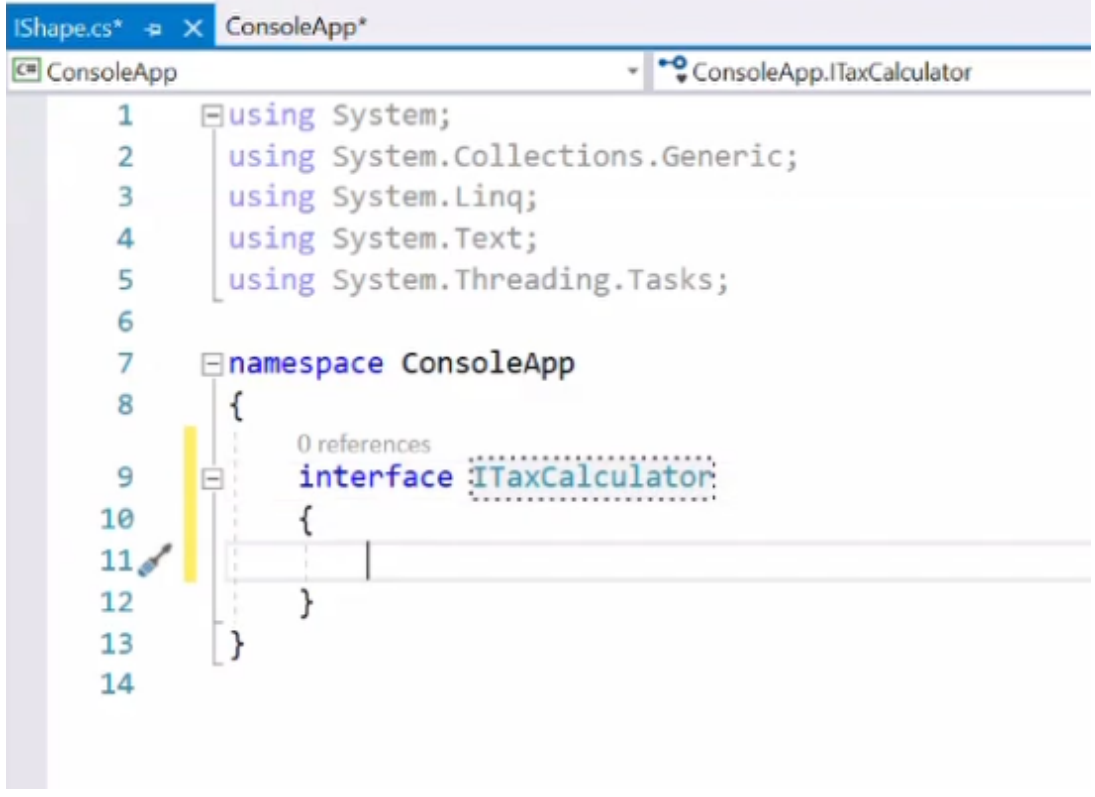
So we make sure that we not use **extends** here, In Java we say extends naa, we'll avoid using that as much as possible

Now whenever you want to create Interface follow this steps


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
ConsoleApp
Debug
Any CPU
Start
Live Share
Server Explorer
Toolbox
ConsoleApp
Application
Build
Build Events
Debug
Resources
Services
Settings
Reference Paths
Signing
Security
Publish
Code Analysis
Configuration: N/A
Platform: N/A
Assembly name:
ConsoleApp
Default namespace:
ConsoleApp
Target framework:
.NET Framework 4.5
Output type:
Console Application
Auto-generate binding redirects
Startup object:
ConsoleApp.InhDemo1
Resources
Specify how application resources will be managed:
Icon and manifest
A manifest determines specific settings for an application. To embed a cust
it to your project and then select it from the list below.
Icon:
(Default Icon)
Manifest:
Embed manifest with default settings
Resource file:
Output
Package Manager Console
Error List
Immediate Window
Ready
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Folder
Container Orchestrator Support...
REST API Client...
Machine Learning Model...
Reference...
Web Reference...
Service Reference...
Connected Service
Analyzer...
Form (Windows Forms)...
User Control (Windows Forms)...
Component...
Class...
New EditorConfig
Build
Rebuild
Clean
Analyze and Code Cleanup
Publish...
Publish as Azure WebJob...
Scope to This
New Solution Explorer View
Build Dependencies
Add
Manage NuGet Packages...
Set as Startup Project
Debug
Initialize Interactive with Project
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename F2
Unload Project
Load Direct Dependencies of Project
Load Entire Dependency Tree of Project
Copy Full Path
Open Folder in File Explorer
Open in Terminal
Properties Alt+Enter
Type here to search
ENG
04:16 PM
22-10-2021


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
ConsoleApp
Debug
Any CPU
Start
Live Share
Add New Item - ConsoleApp
Application
Build
Build Events
Debug
Resources
Services
Settings
Reference Paths
Signing
Security
Publish
Code Analysis
Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
MySQL
SQL Server
Online
Sort by: Default
Search (Ctrl+E)
Class Visual C# Items
Interface Visual C# Items
Form (Windows Forms) Visual C# Items
User Control (Windows Forms) Visual C# Items
Component Class Visual C# Items
User Control (WPF) Visual C# Items
About Box (Windows Forms) Visual C# Items
ADO.NET Entity Data Model Visual C# Items
Application Configuration File Visual C# Items
Application Manifest File (Windows Only) Visual C# Items
Type: Visual C# Items
An empty interface definition
Name: Class1.cs
Add
Cancel
Output
Package Manager Console
Error List
Immediate Window
Ready
Add to Source Control
Type here to search
ENG
04:16 PM
22-10-2021

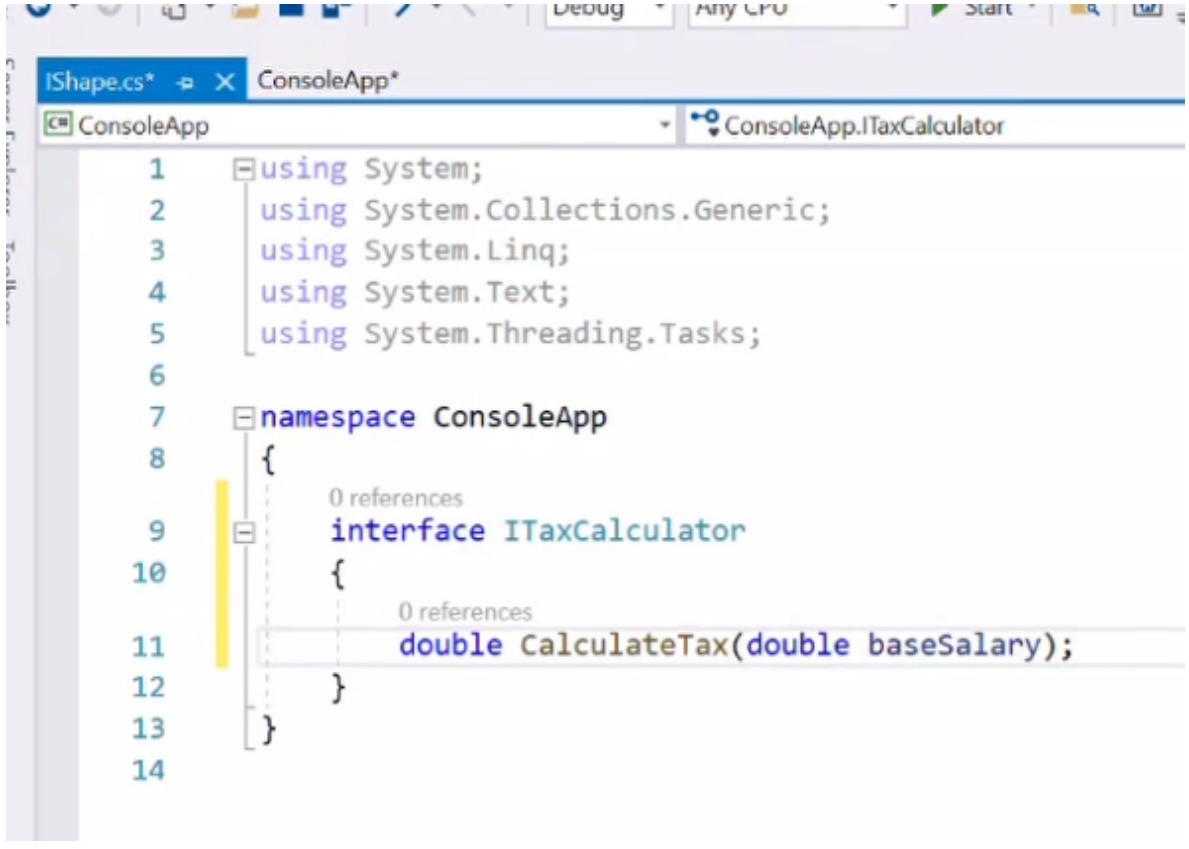
and whenever you create Interface it's name should always starts with I



this is the taxes

Tax Slab(₹)	Old Tax Rates	New Tax Rates
0 – 2,50,000	0%	0%
2,50,000 – 5,00,000	5%	5%
5,00,000 – 7,50,000	20%	10%
7,50,000 – 10,00,000	20%	15%
10,00,000 – 12,50,000	30%	20%
12,50,000 – 15,00,000	30%	25%
15,00,000 & above	30%	30%

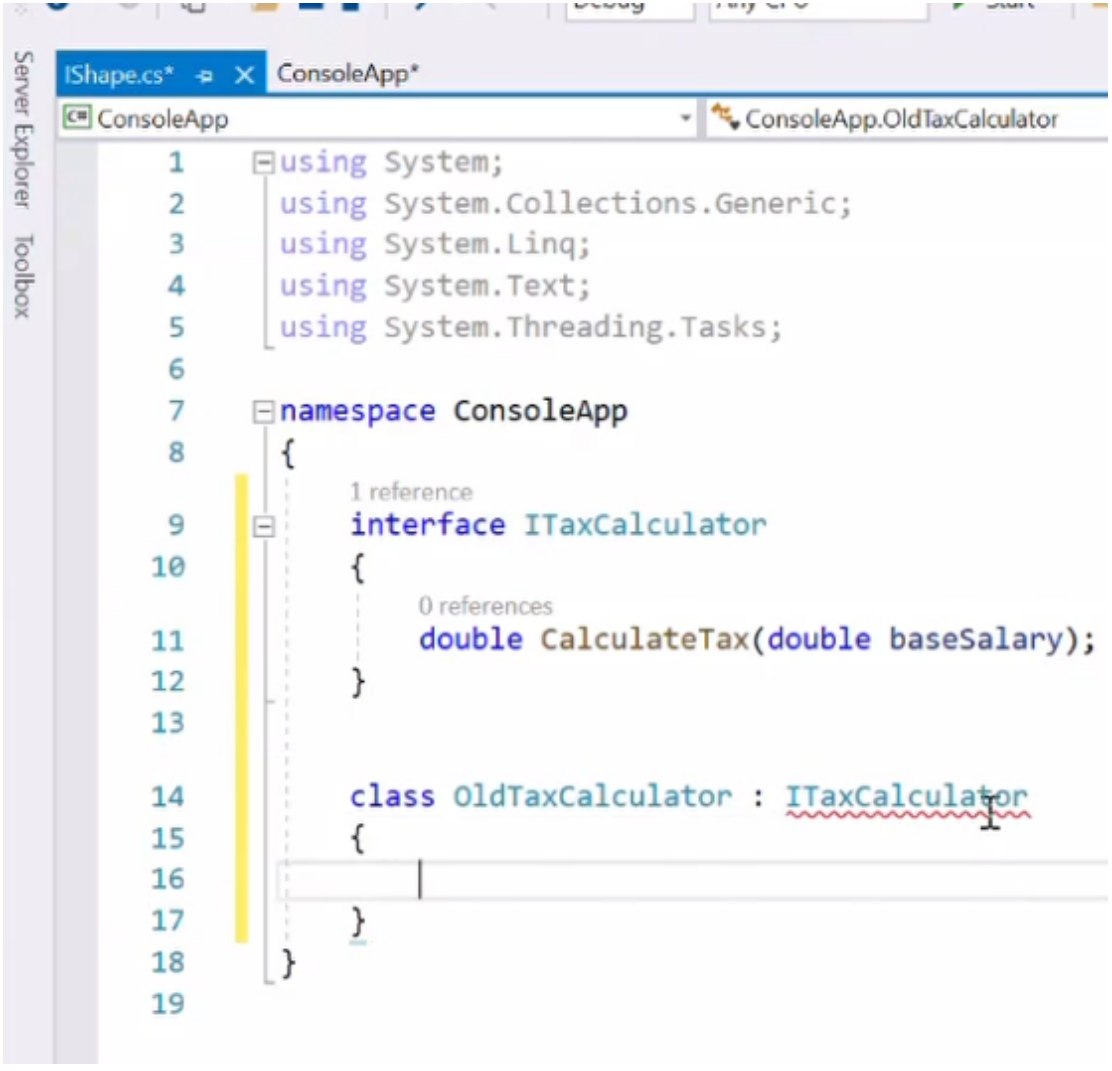
and I want to calculate the tax
So to calculate the tax you just give me the base salary



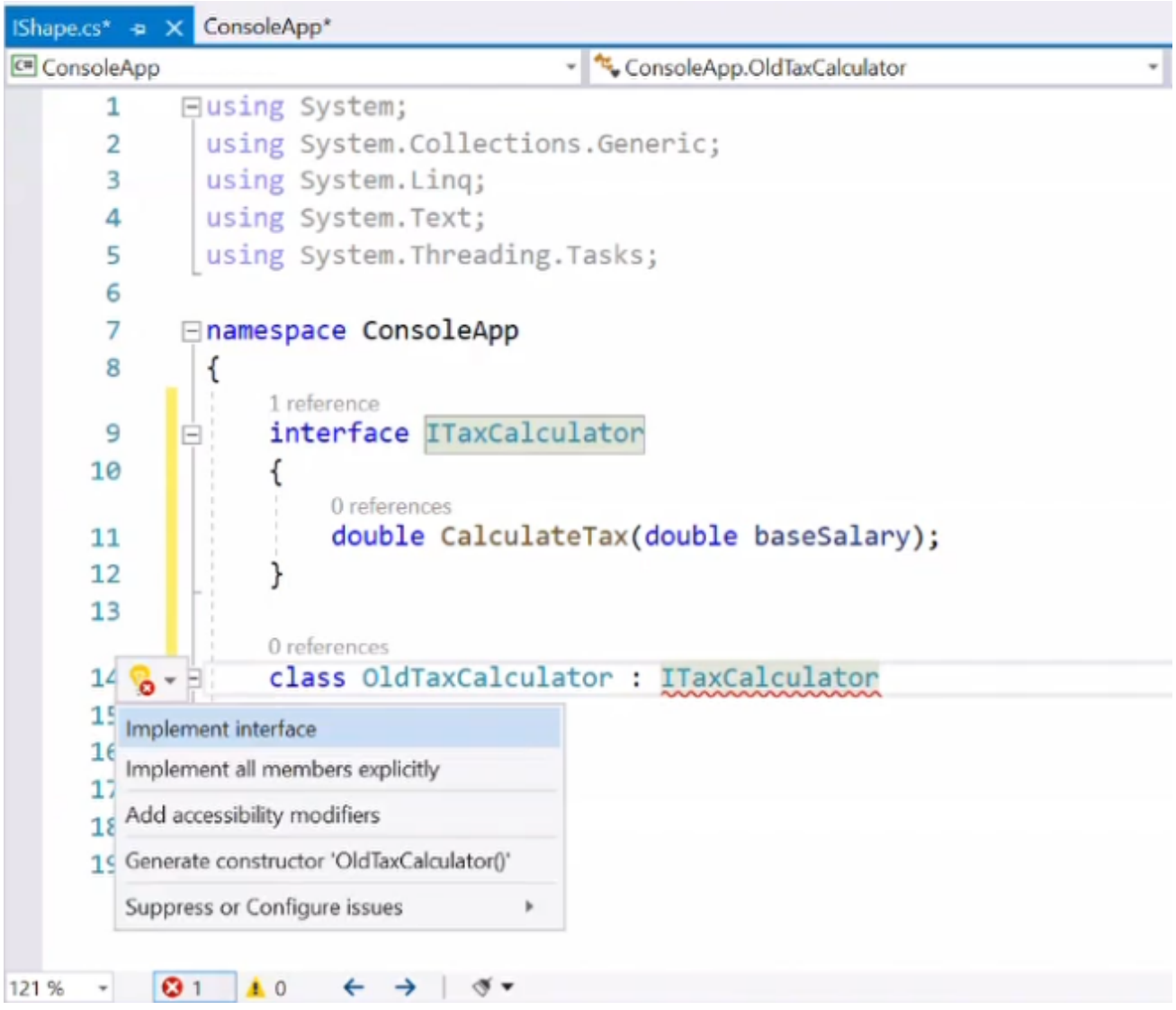
This is the feature, what I'm trying to identify in my taxcalculator
So what my TaxCalculator should do,
My TaxCalculator Should always adhere to a particular rule or a feature called as CalculateTax

just like mobile phone, What is the basic feature of mobile phone
---- it should make call , recieve call , send text message after these 3 feature other feature will come

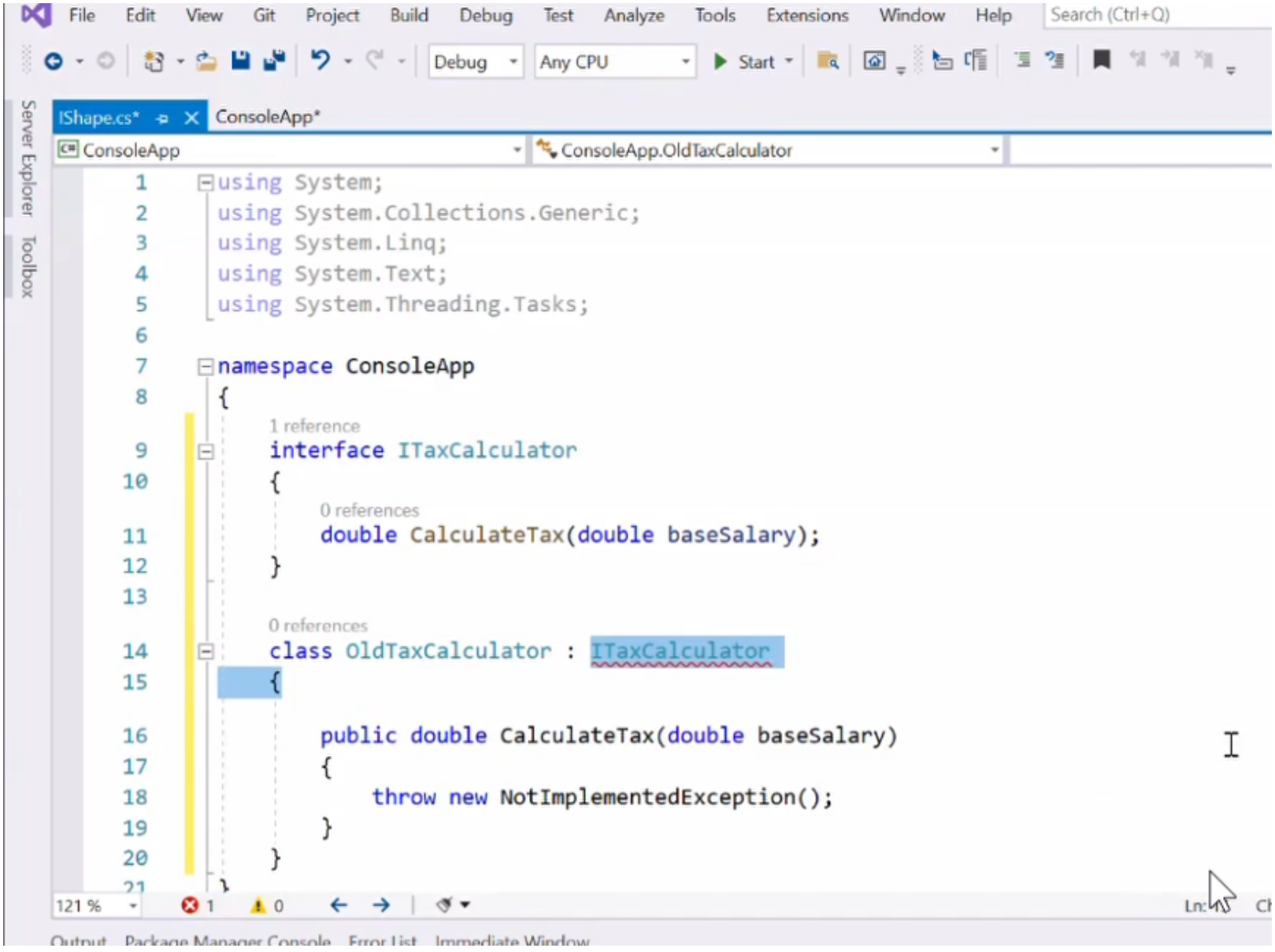
I'll say OldTaxCalculator which inherit ITaxCalculator



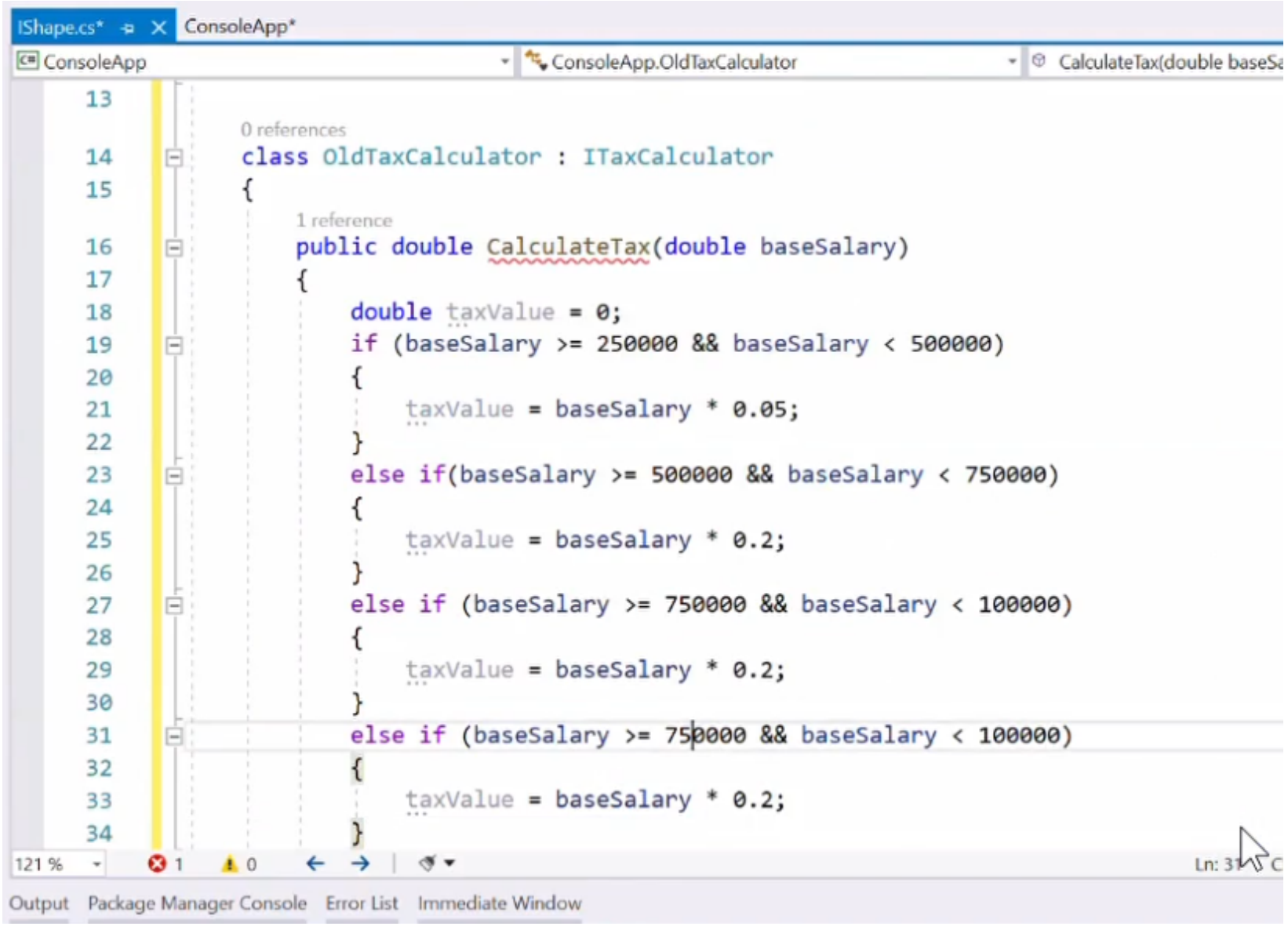
now as soon as I press CTRL + .

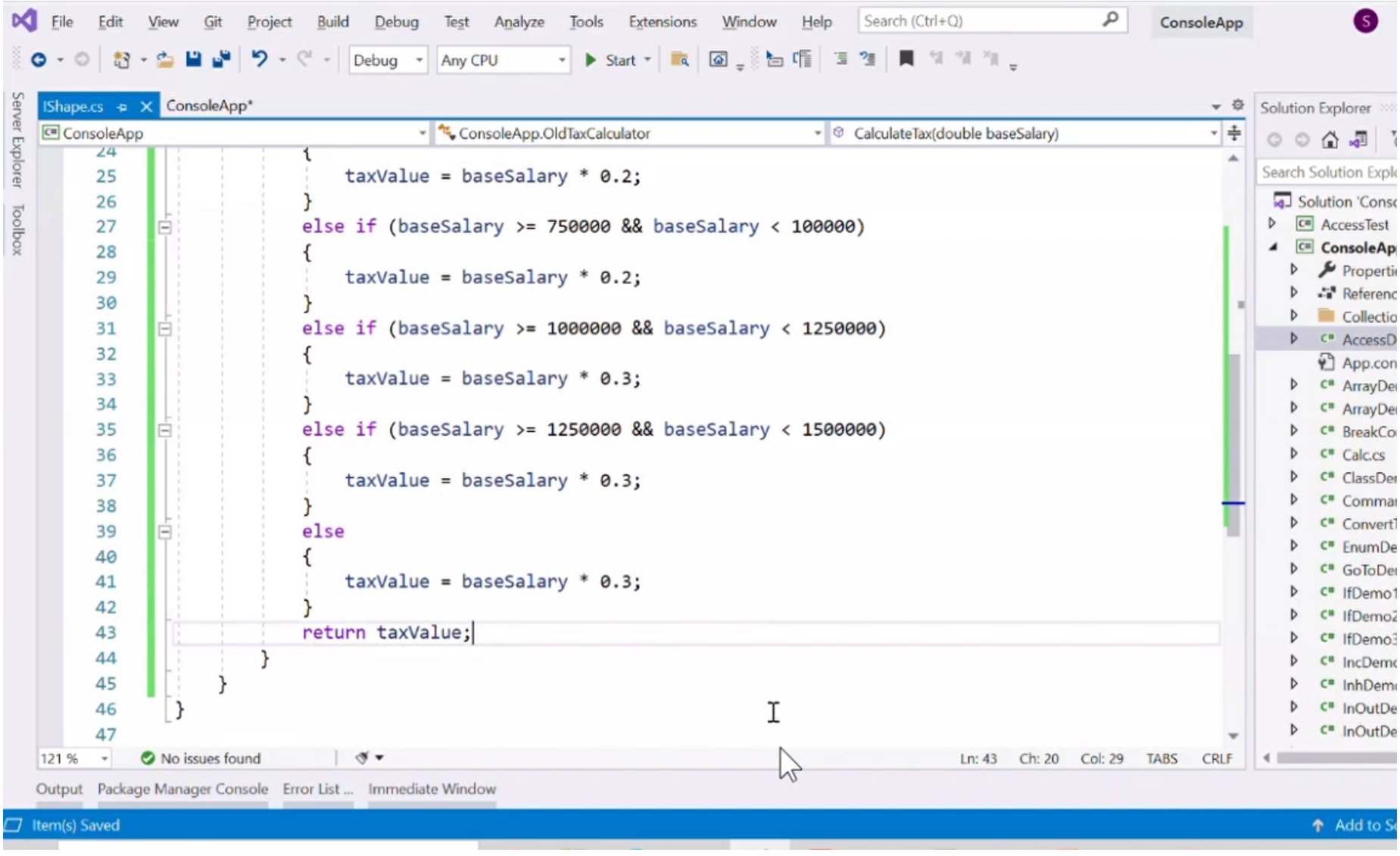


and click on Implement Interface

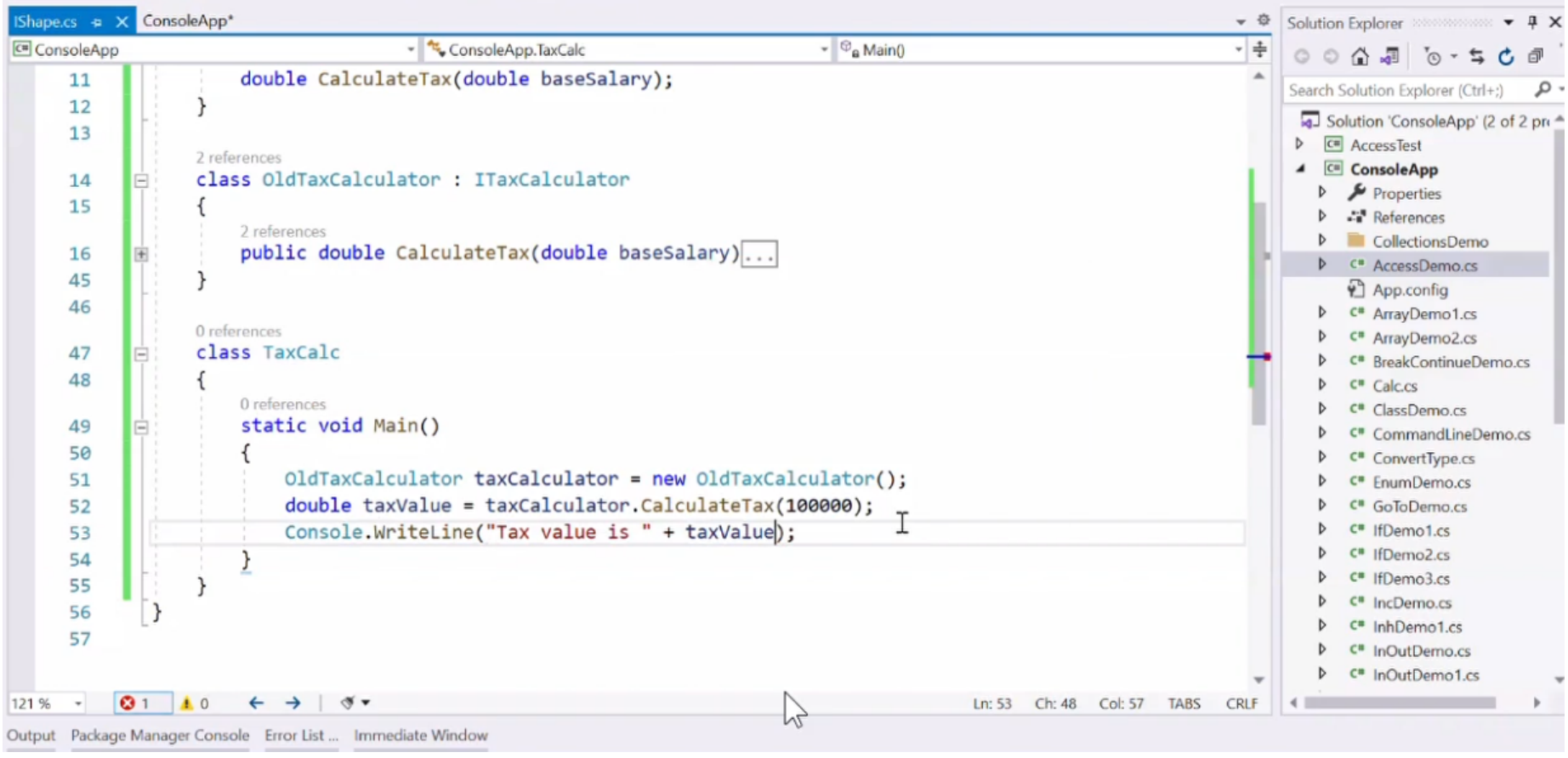

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Any CPU
Start
IShape.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.OldTaxCalculator
Server Explorer
Toolbox
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
1 reference
interface ITaxCalculator
{
0 references
double CalculateTax(double baseSalary);
}
0 references
class OldTaxCalculator : ITaxCalculator
{
public double CalculateTax(double baseSalary)
{
throw new NotImplementedException();
}
}
121 %
1
0
Ln:
Output
Package Manager Console
Error List
Immediate Window

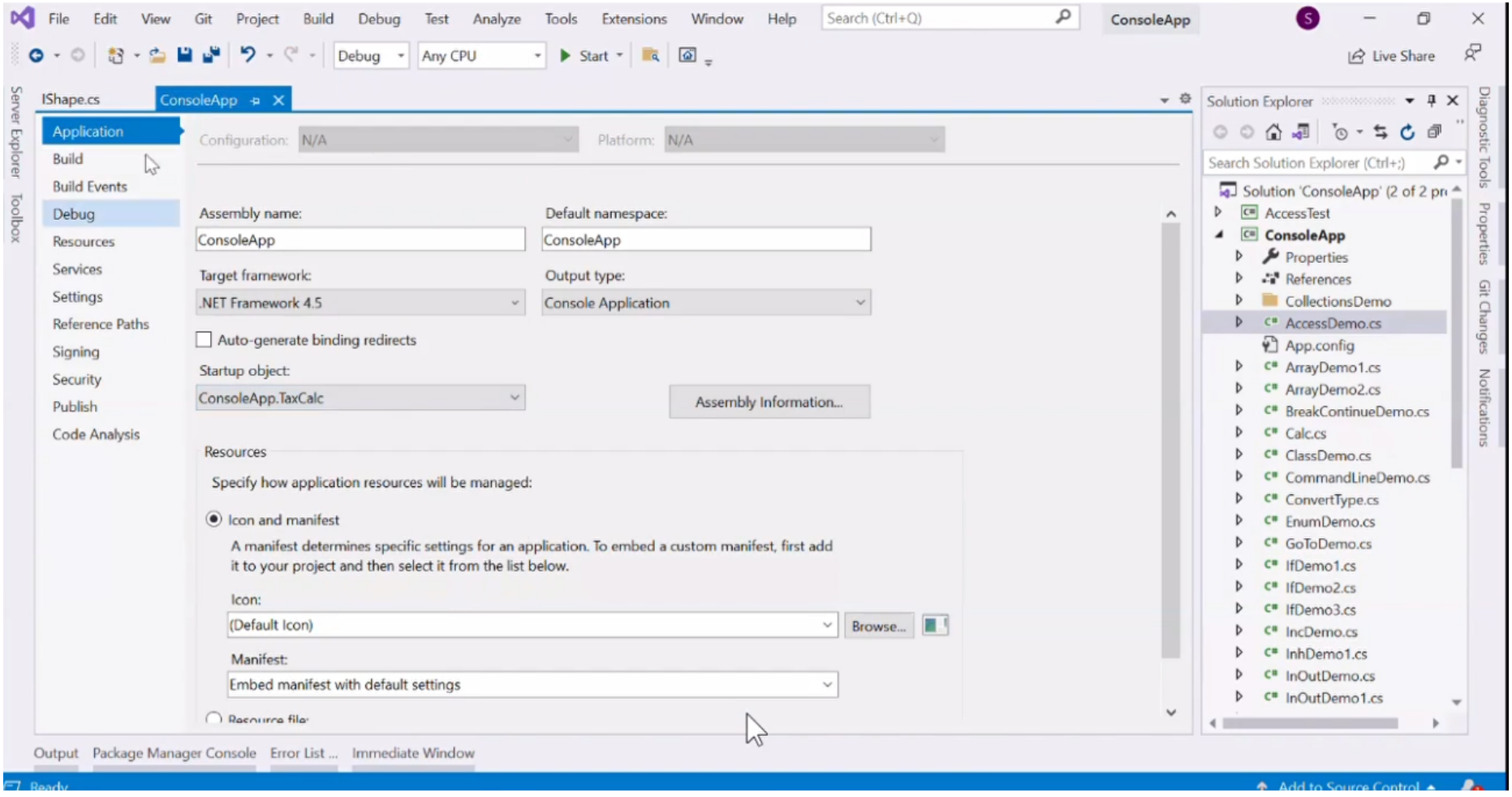
i get this
and according to the taxation chart i'll create the program


IShape.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.OldTaxCalculator
CalculateTax(double baseSa
0 references
class OldTaxCalculator : ITaxCalculator
{
1 reference
public double CalculateTax(double baseSalary)
{
double taxValue = 0;
if (baseSalary >= 250000 && baseSalary < 500000)
{
taxValue = baseSalary * 0.05;
}
else if(baseSalary >= 500000 && baseSalary < 750000)
{
taxValue = baseSalary * 0.2;
}
else if (baseSalary >= 750000 && baseSalary < 100000)
{
taxValue = baseSalary * 0.2;
}
else if (baseSalary >= 750000 && baseSalary < 100000)
{
taxValue = baseSalary * 0.2;
}
121 %
1
0
Output
Package Manager Console
Error List
Immediate Window

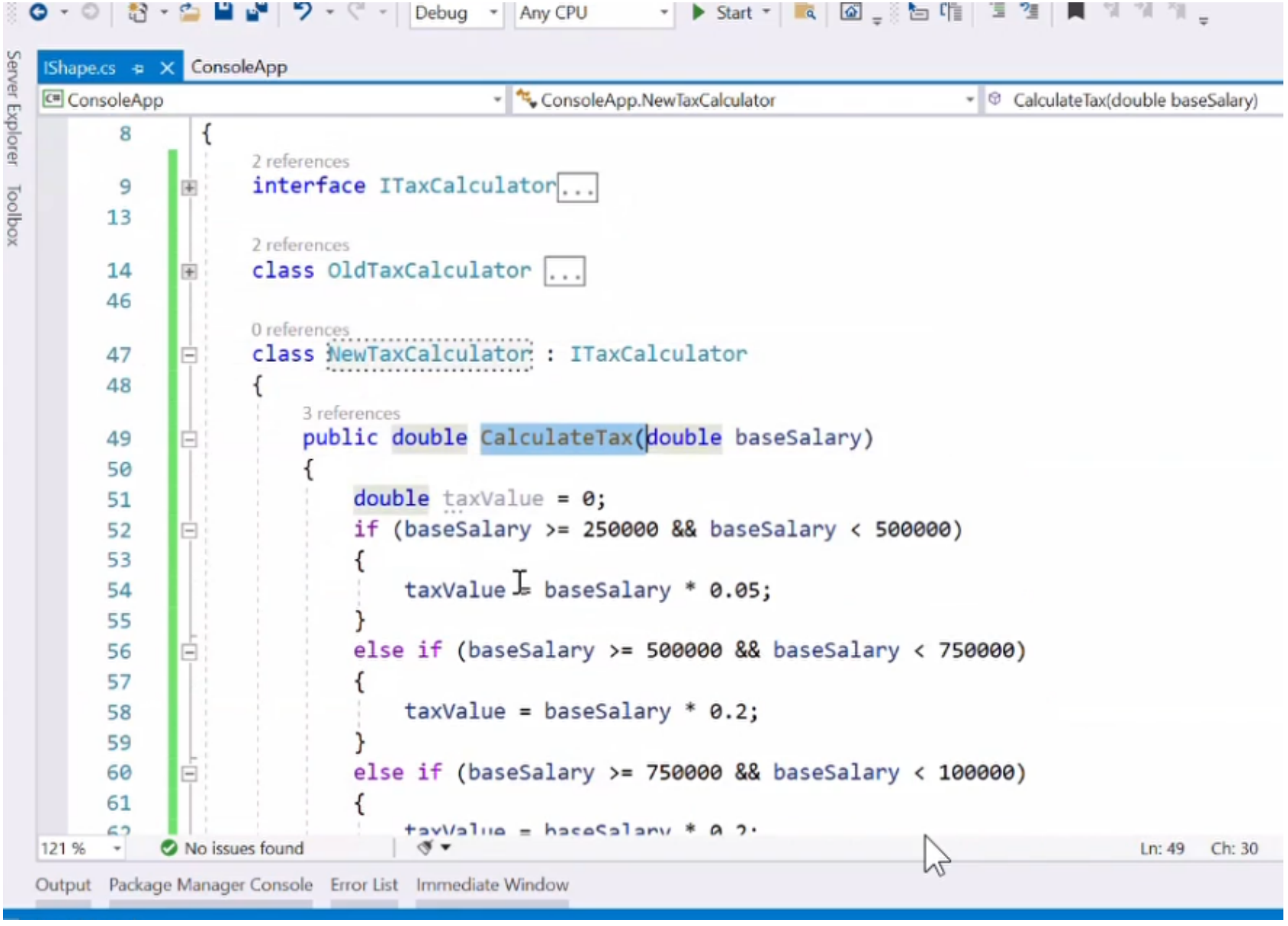

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
ConsoleApp
Debug
Any CPU
Start
IShape.cs
ConsoleApp*
ConsoleApp
ConsoleApp.OldTaxCalculator
CalculateTax(double baseSalary)
{
taxValue = baseSalary * 0.2;
}
else if (baseSalary >= 750000 && baseSalary < 100000)
{
taxValue = baseSalary * 0.2;
}
else if (baseSalary >= 1000000 && baseSalary < 1250000)
{
taxValue = baseSalary * 0.3;
}
else if (baseSalary >= 1250000 && baseSalary < 1500000)
{
taxValue = baseSalary * 0.3;
}
else
{
taxValue = baseSalary * 0.3;
}
return taxValue;
}
}
}
Solution Explorer
Search Solution Explo
Solution 'Conso
AccessTest
ConsoleAp
Properti
Referenc
Collectio
AccessD
App.con
ArrayDe
ArrayDe
BreakCo
Calc.cs
ClassDe
Comma
Convert
EnumDe
GoToDe
IfDemo1
IfDemo2
IfDemo3
IncDemo
InhDemo
InOutDe
InOutDe
121 %
No issues found
Ln: 43 Ch: 20 Col: 29 TABS CRLF
Output
Package Manager Console
Error List ...
Immediate Window
Item(s) Saved
Add to S

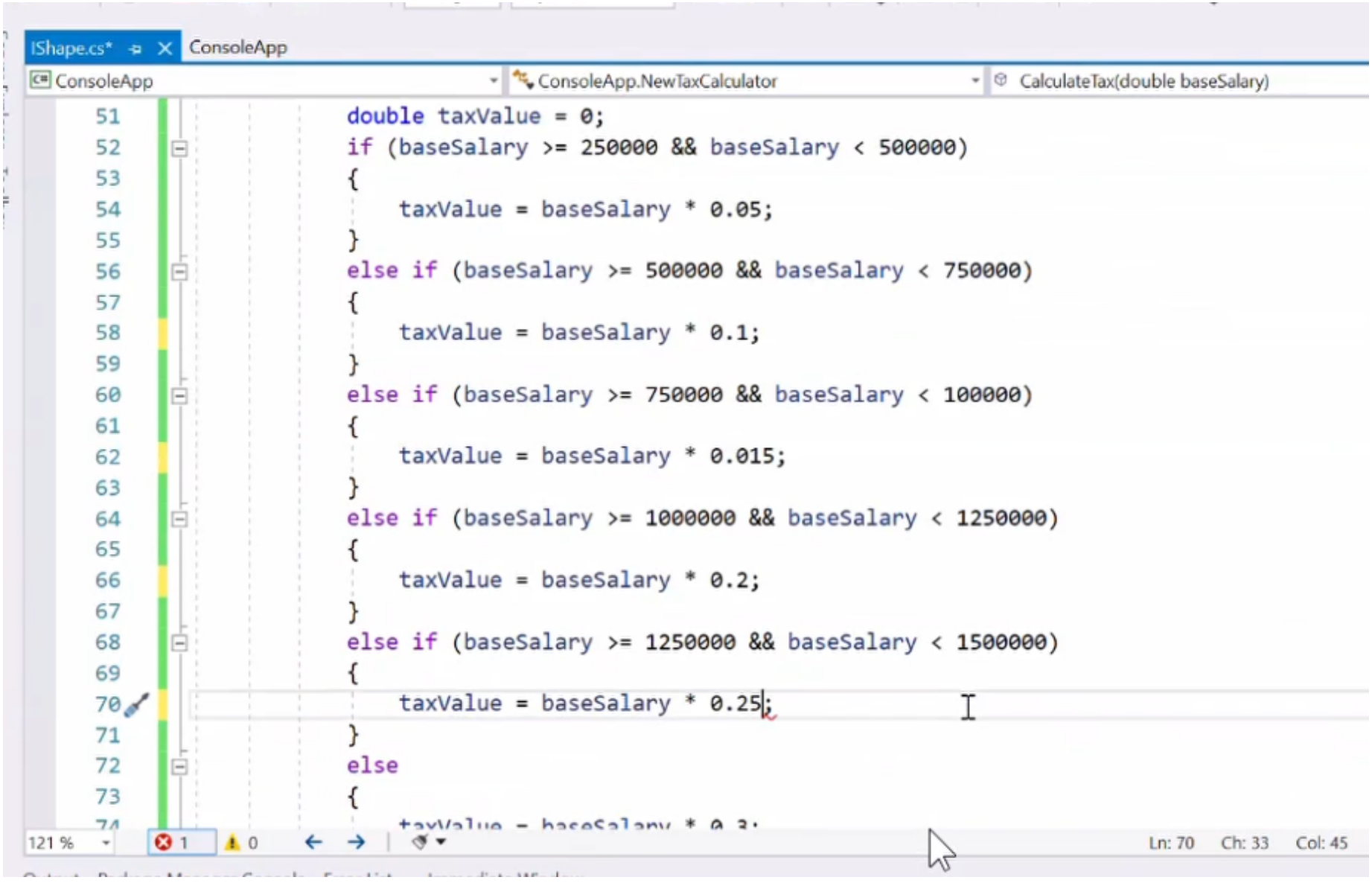
now this is for old tax calculation

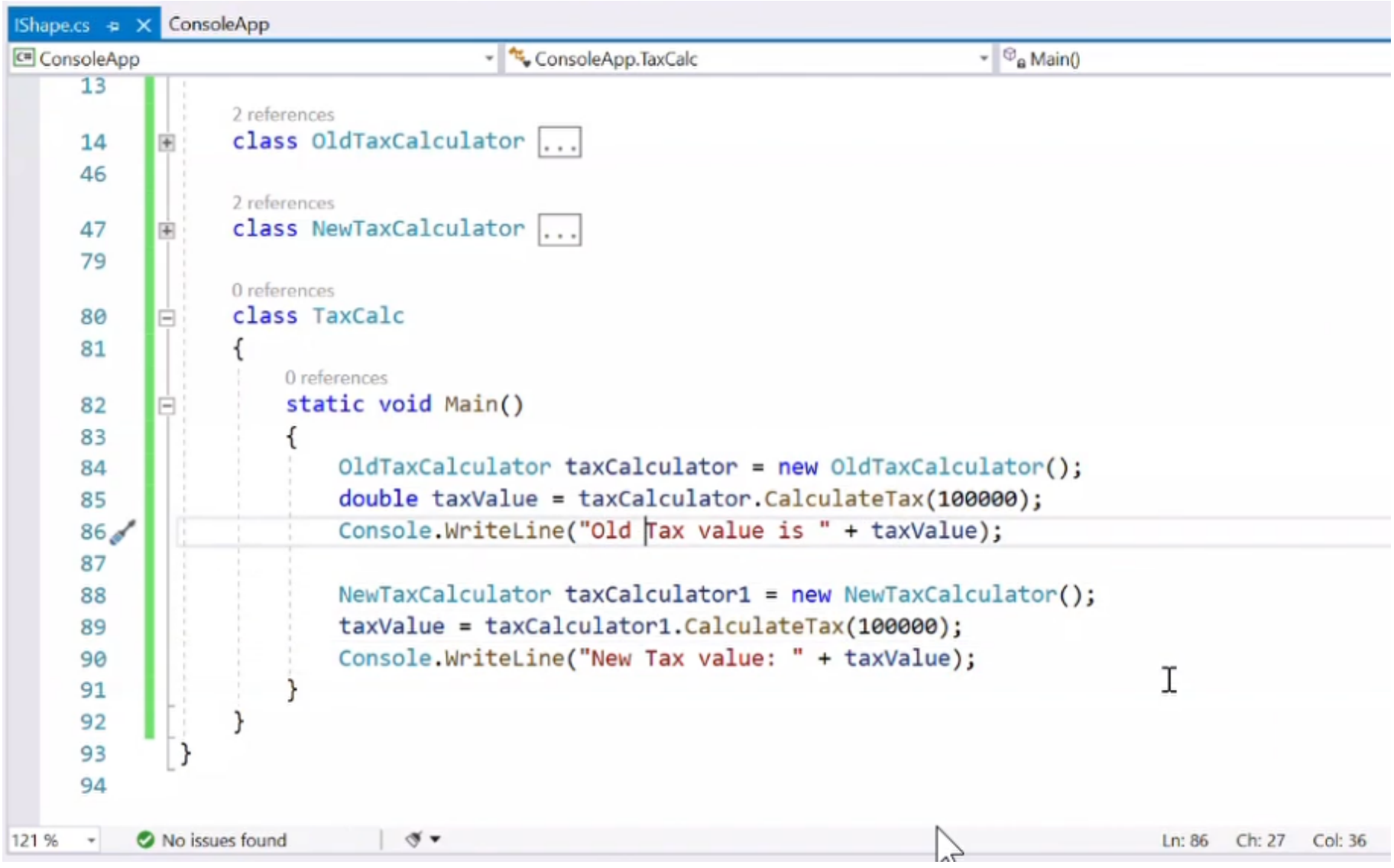


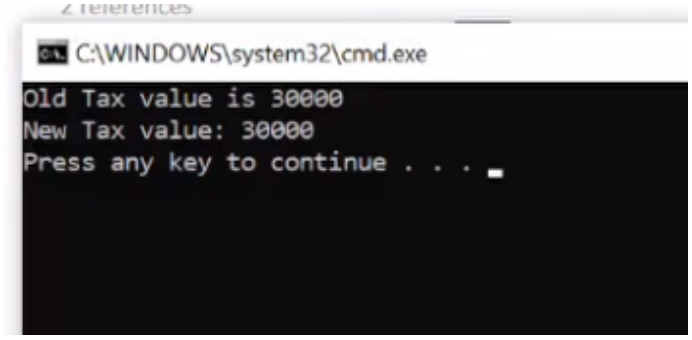

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
ConsoleApp
Debug
Any CPU
Start
Live Share
IShape.cs
ConsoleApp
Application
Build
Build Events
Debug
Resources
Services
Settings
Reference Paths
Signing
Security
Publish
Code Analysis
Configuration: N/A
Platform: N/A
Assembly name:
ConsoleApp
Default namespace:
ConsoleApp
Target framework:
.NET Framework 4.5
Output type:
Console Application
Auto-generate binding redirects
Startup object:
ConsoleApp.TaxCalc
Assembly Information...
Resources
Specify how application resources will be managed:
Icon and manifest
A manifest determines specific settings for an application. To embed a custom manifest, first add it to your project and then select it from the list below.
Icon:
(Default Icon)
Browse...
Manifest:
Embed manifest with default settings
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
InhDemo1.cs
InOutDemo.cs
InOutDemo1.cs
Server Explorer
Toolbox
Diagnostic Tools
Properties
Git Changes
Notifications
Output
Package Manager Console
Error List ...
Immediate Window
Ready
Add to Source Control

similarly we'll write for NewTaxCalculator

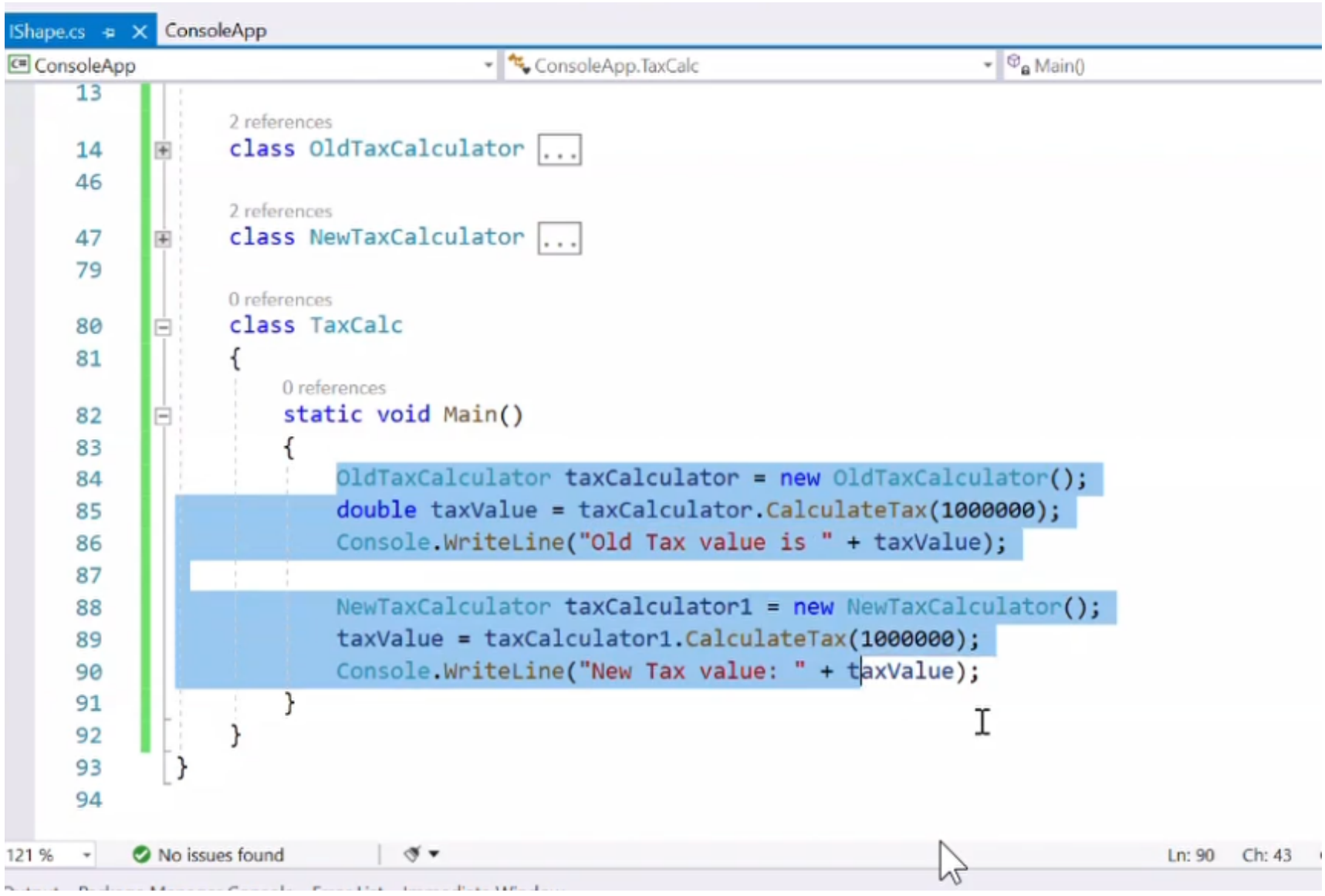

Debug
Any CPU
Start
IShape.cs
ConsoleApp
ConsoleApp
ConsoleApp.NewTaxCalculator
CalculateTax(double baseSalary)
Server Explorer
Toolbox
8 {
2 references
9 interface ITaxCalculator...
13
2 references
14 class OldTaxCalculator ...
46
0 references
47 class NewTaxCalculator : ITaxCalculator
48 {
3 references
49 public double CalculateTax(double baseSalary)
50 {
51 double taxValue = 0;
52 if (baseSalary >= 250000 && baseSalary < 500000)
53 {
54 taxValue = baseSalary * 0.05;
55 }
56 else if (baseSalary >= 500000 && baseSalary < 750000)
57 {
58 taxValue = baseSalary * 0.2;
59 }
60 else if (baseSalary >= 750000 && baseSalary < 100000)
61 {
121 %
No issues found
Ln: 49
Ch: 30
Output
Package Manager Console
Error List
Immediate Window


IShape.cs*
ConsoleApp
ConsoleApp
ConsoleApp.NewTaxCalculator
CalculateTax(double baseSalary)
51 double taxValue = 0;
52 if (baseSalary >= 250000 && baseSalary < 500000)
53 {
54 taxValue = baseSalary * 0.05;
55 }
56 else if (baseSalary >= 500000 && baseSalary < 750000)
57 {
58 taxValue = baseSalary * 0.1;
59 }
60 else if (baseSalary >= 750000 && baseSalary < 100000)
61 {
62 taxValue = baseSalary * 0.015;
63 }
64 else if (baseSalary >= 1000000 && baseSalary < 1250000)
65 {
66 taxValue = baseSalary * 0.2;
67 }
68 else if (baseSalary >= 1250000 && baseSalary < 1500000)
69 {
70 taxValue = baseSalary * 0.25;
71 }
72 else
73 {
121 %
1
0
Ln: 70
Ch: 33
Col: 45

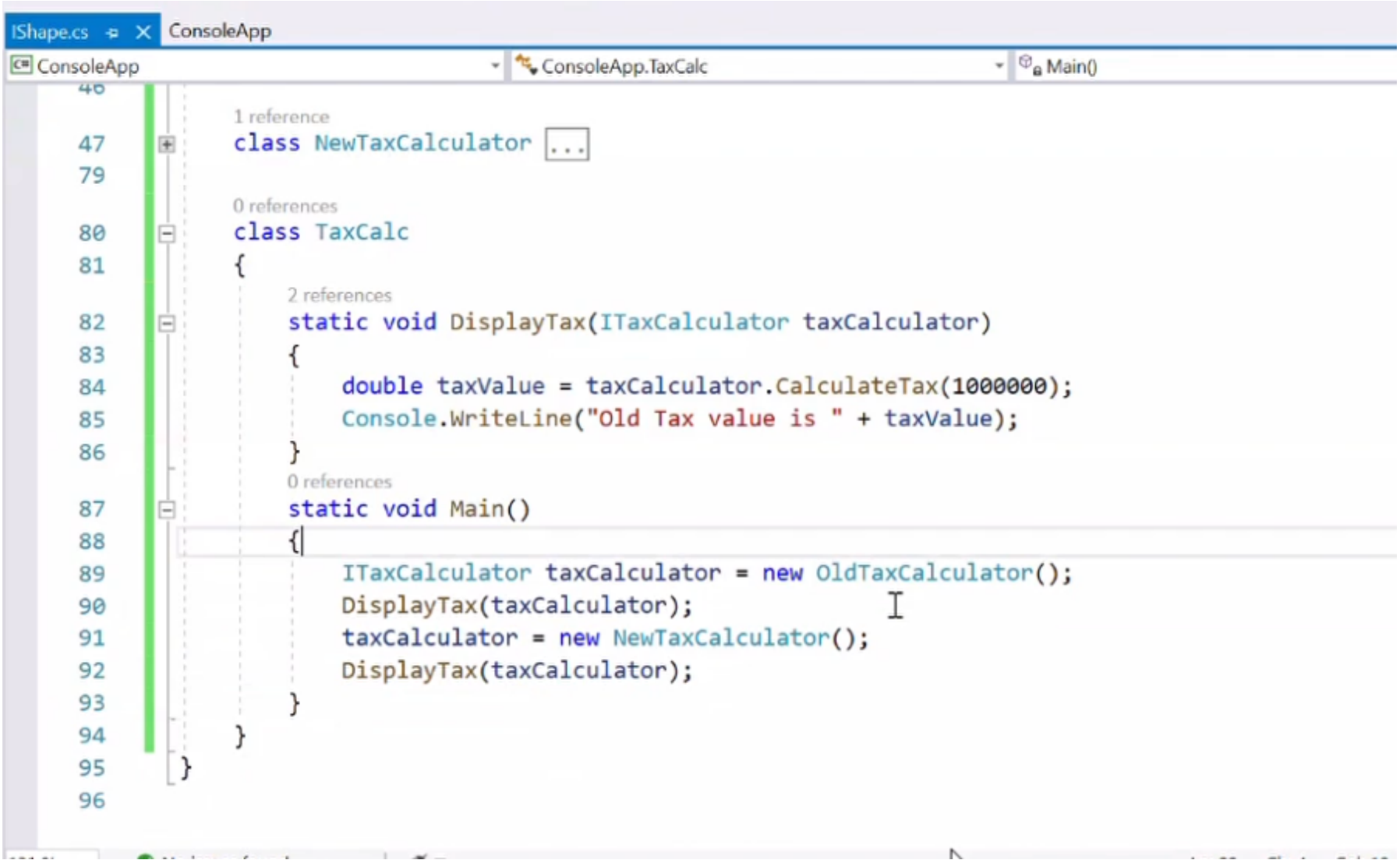

IShape.cs
ConsoleApp
ConsoleApp
ConsoleApp.TaxCalc
Main()
2 references
class OldTaxCalculator ...
2 references
class NewTaxCalculator ...
0 references
class TaxCalc
{
0 references
static void Main()
{
OldTaxCalculator taxCalculator = new OldTaxCalculator();
double taxValue = taxCalculator.CalculateTax(100000);
Console.WriteLine("Old Tax value is " + taxValue);
NewTaxCalculator taxCalculator1 = new NewTaxCalculator();
taxValue = taxCalculator1.CalculateTax(100000);
Console.WriteLine("New Tax value: " + taxValue);
}
}
}
121 %
No issues found
Ln: 86
Ch: 27
Col: 36



Now can you see that this is repeated code


IShape.cs
ConsoleApp
ConsoleApp
ConsoleApp.TaxCalc
Main()
2 references
class OldTaxCalculator ...
2 references
class NewTaxCalculator ...
0 references
class TaxCalc
{
0 references
static void Main()
{
OldTaxCalculator taxCalculator = new OldTaxCalculator();
double taxValue = taxCalculator.CalculateTax(1000000);
Console.WriteLine("Old Tax value is " + taxValue);
NewTaxCalculator taxCalculator1 = new NewTaxCalculator();
taxValue = taxCalculator1.CalculateTax(1000000);
Console.WriteLine("New Tax value: " + taxValue);
}
}
}
121 %
No issues found
Ln: 90
Ch: 43

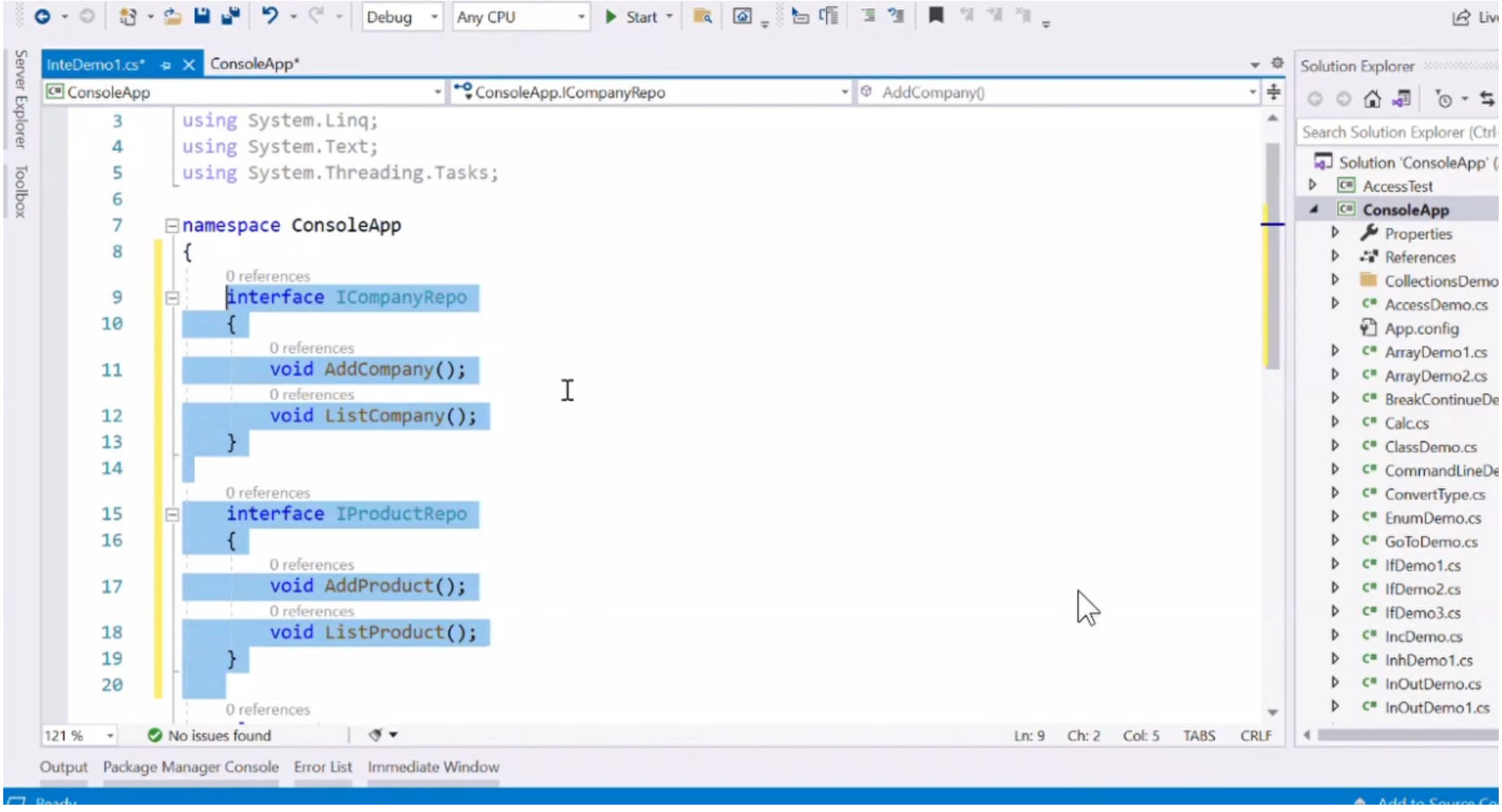
so you can always call this display method
so declaration and printing i'll write it there



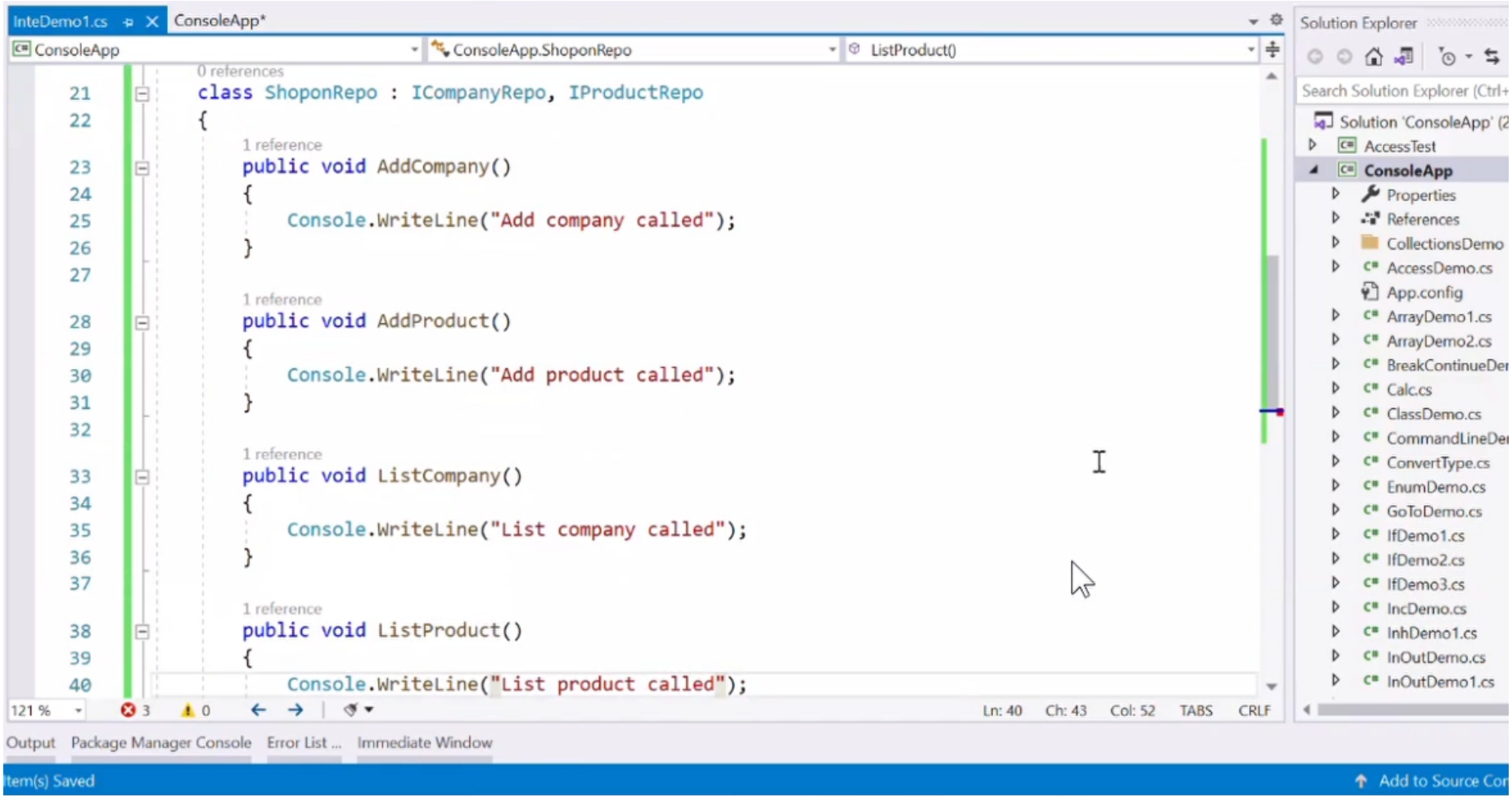
So Interface are mainly use to set the rule

and Implementation class will adhere to that particular rule

Suppose you are writing program for this company and order
this will help me to add and fetch the company details
similarly we have IProduct interface
to add and get the product details

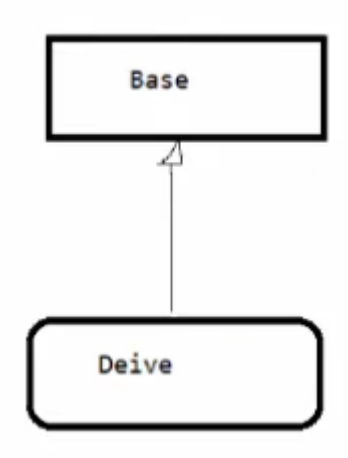


then we write ShoponRepo class and inherit both the interface here


InteDemo1.cs
ConsoleApp*
ConsoleApp
ConsoleApp.ShoponRepo
ListProduct()
0 references
class ShoponRepo : ICompanyRepo, IProductRepo
{
1 reference
public void AddCompany()
{
Console.WriteLine("Add company called");
}
1 reference
public void AddProduct()
{
Console.WriteLine("Add product called");
}
1 reference
public void ListCompany()
{
Console.WriteLine("List company called");
}
1 reference
public void ListProduct()
{
Console.WriteLine("List product called");
121 %
3
0
Ln: 40
Ch: 43
Col: 52
TABS
CRLF
Output
Package Manager Console
Error List ...
Immediate Window
Item(s) Saved
Add to Source Co
Solution Explorer
Search Solution Explorer (Ctrl+
Solution 'ConsoleApp' (2
AccessTest
ConsoleApp
Properties
References
CollectionsDemo
AccessDemo.cs
App.config
ArrayDemo1.cs
ArrayDemo2.cs
BreakContinueDe
Calc.cs
ClassDemo.cs
CommandLineDe
ConvertType.cs
EnumDemo.cs
GoToDemo.cs
IfDemo1.cs
IfDemo2.cs
IfDemo3.cs
IncDemo.cs
InhDemo1.cs
InOutDemo.cs
InOutDemo1.cs

What are the different types of Inheritance we have

1. Single Inheritance - 1 base class and 1 Derived class

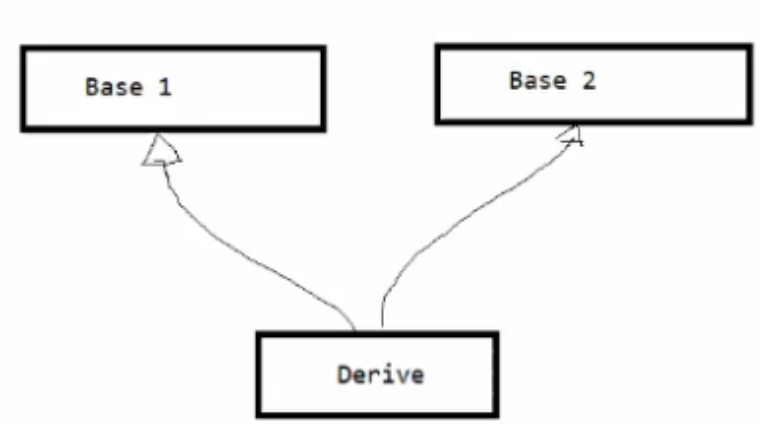


2.

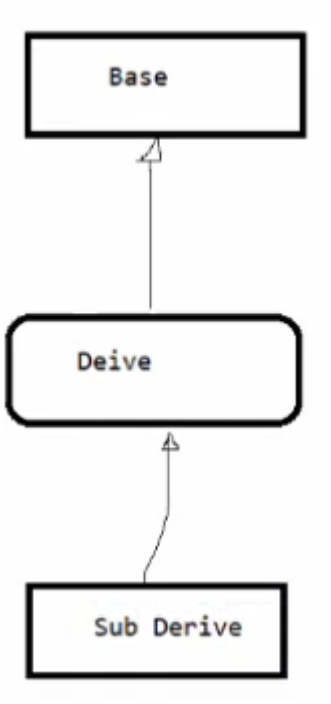
Multiple Inheritance - 2 base class and 1 Derived class (**Java and C# dosen't support Multiple Inheritance**) **if you have to do something like this, you have to achieve this through Interface**

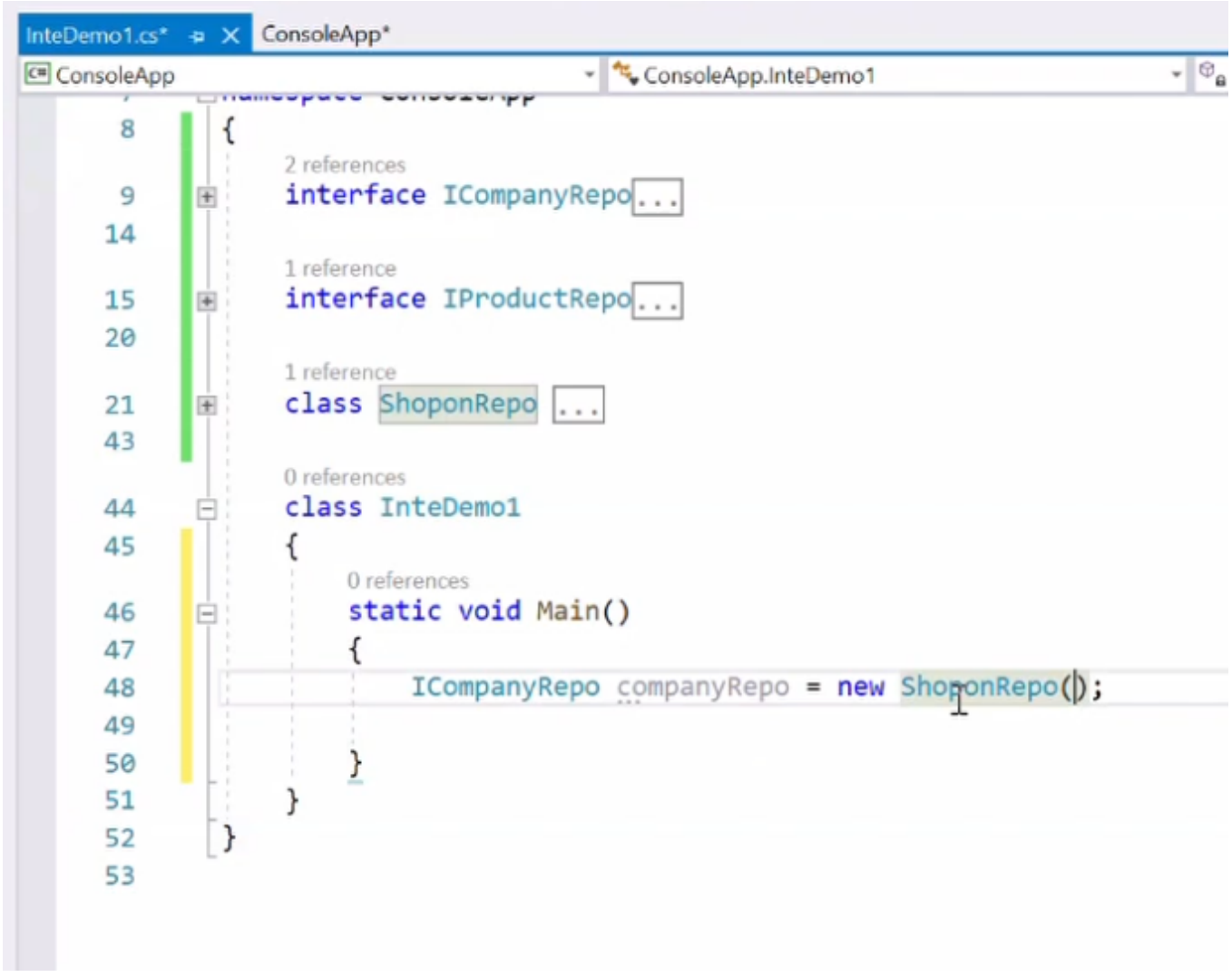
that's what we are doing here 1 class with 2 interfaces

2.

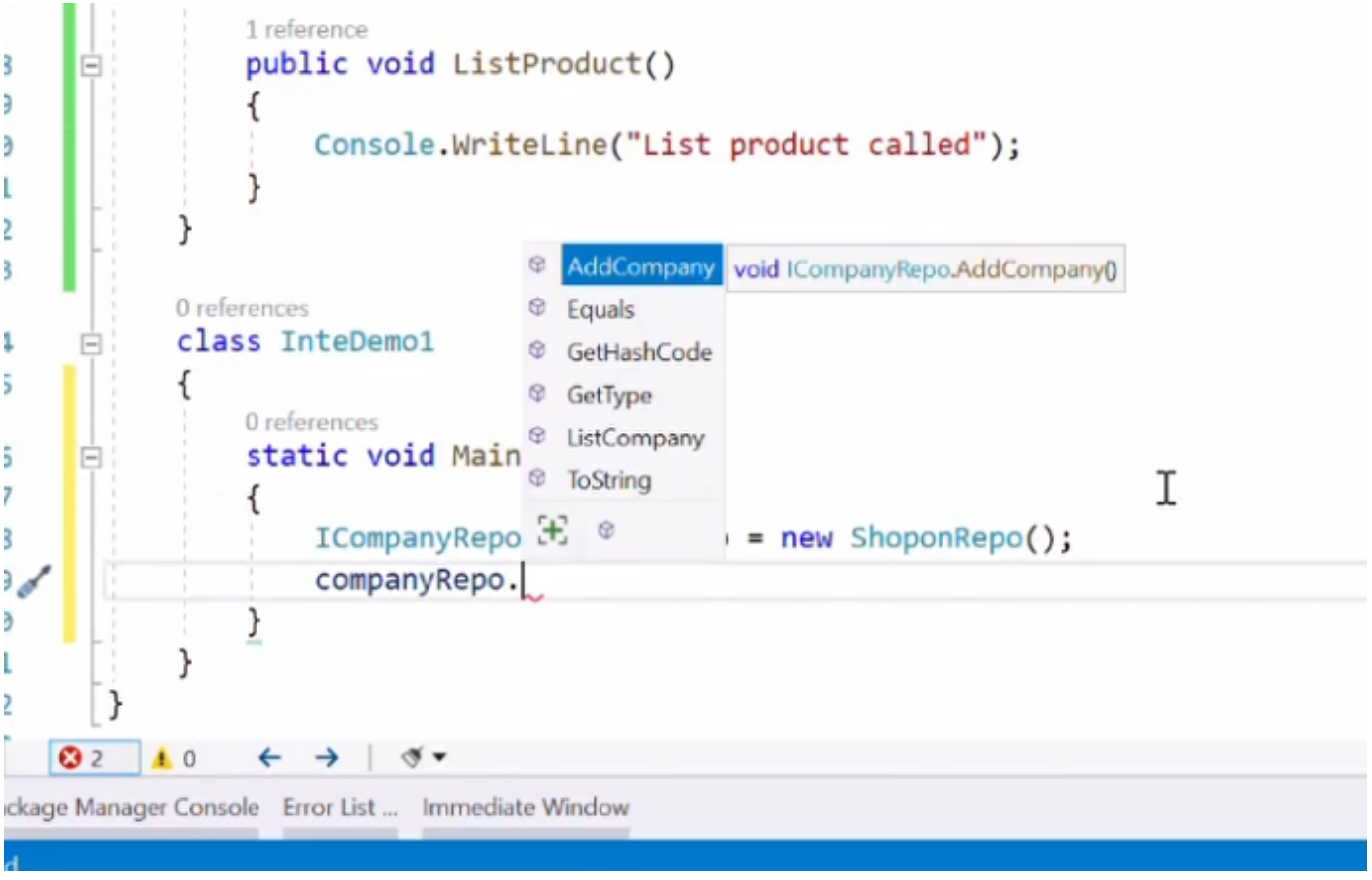


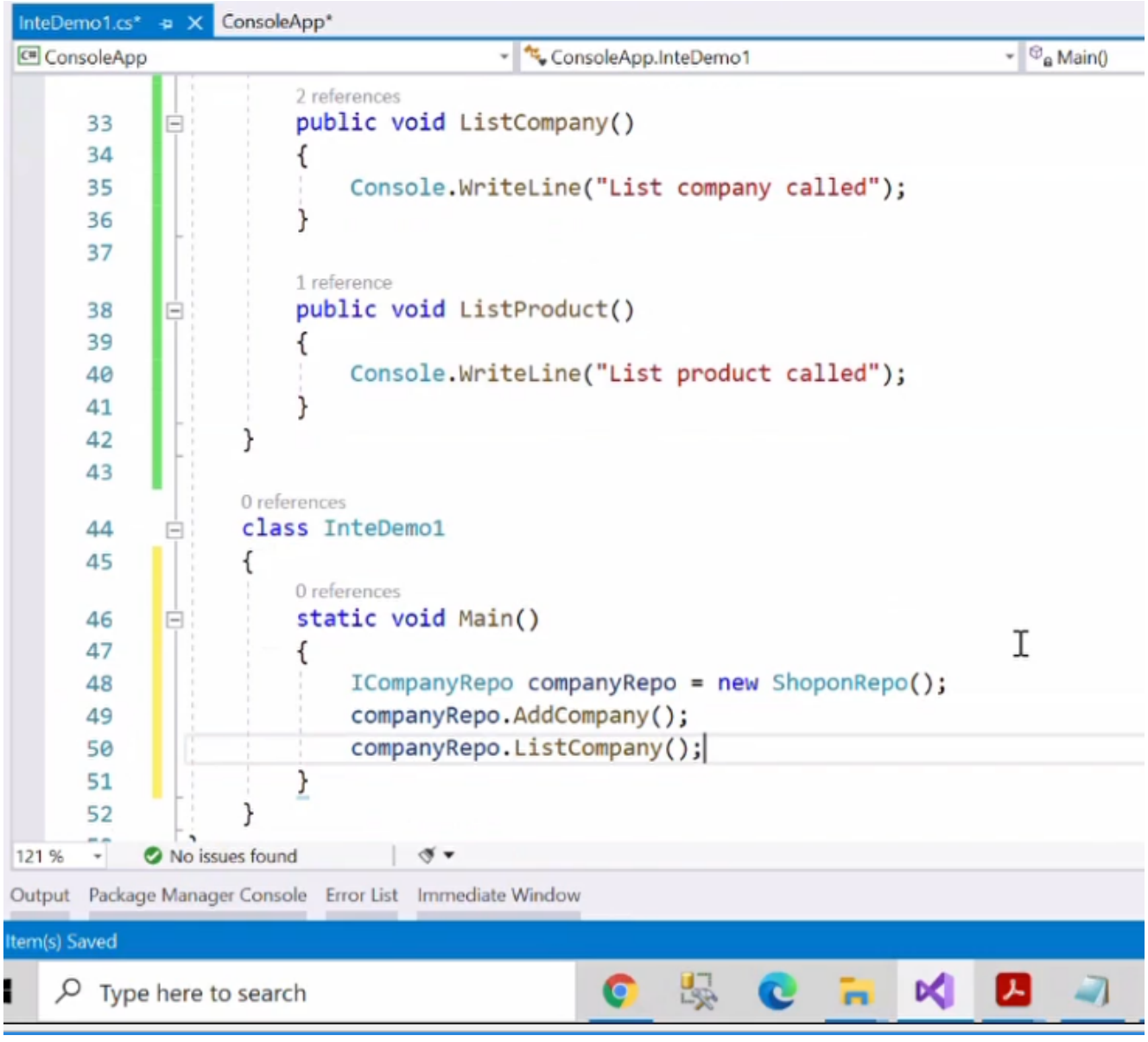
3. Multilevel Inheritance - 1 base class, 1 derived class, 1 derived class





here when i say companyrepo. then what are the method which it should expose as I'm pointing to shopon here but I'm creating reference of ICompanyRepo


1 reference
public void ListProduct()
{
Console.WriteLine("List product called");
}
}
0 references
class InteDemo1
{
0 references
static void Main
{
ICompanyRepo = new ShoponRepo();
companyRepo.
}
}
}
AddCompany
void ICompanyRepo.AddCompany()
Equals
GetHashCode
GetType
ListCompany
ToString
2
0
ckage Manager Console
Error List ...
Immediate Window

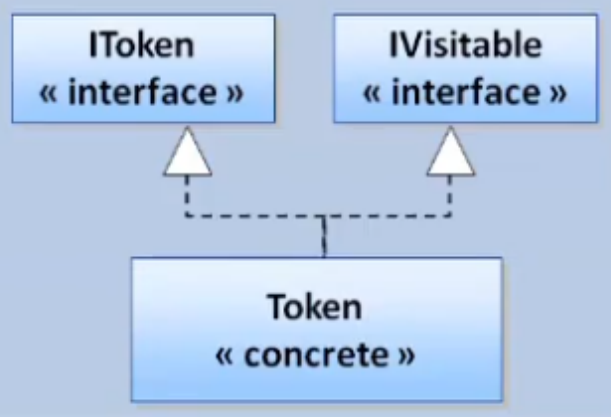

InteDemo1.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.InteDemo1
Main()
2 references
public void ListCompany()
{
Console.WriteLine("List company called");
}
1 reference
public void ListProduct()
{
Console.WriteLine("List product called");
}
}
0 references
class InteDemo1
{
0 references
static void Main()
{
ICompanyRepo companyRepo = new ShoponRepo();
companyRepo.AddCompany();
companyRepo.ListCompany();
}
}
121 %
No issues found
Output
Package Manager Console
Error List
Immediate Window
Item(s) Saved
Type here to search

Where is AddProduct and ListProduct ??
It will not bother, this is the power of Interface

Implementing Multiple Interfaces in a class

- A class can implement zero or more interfaces

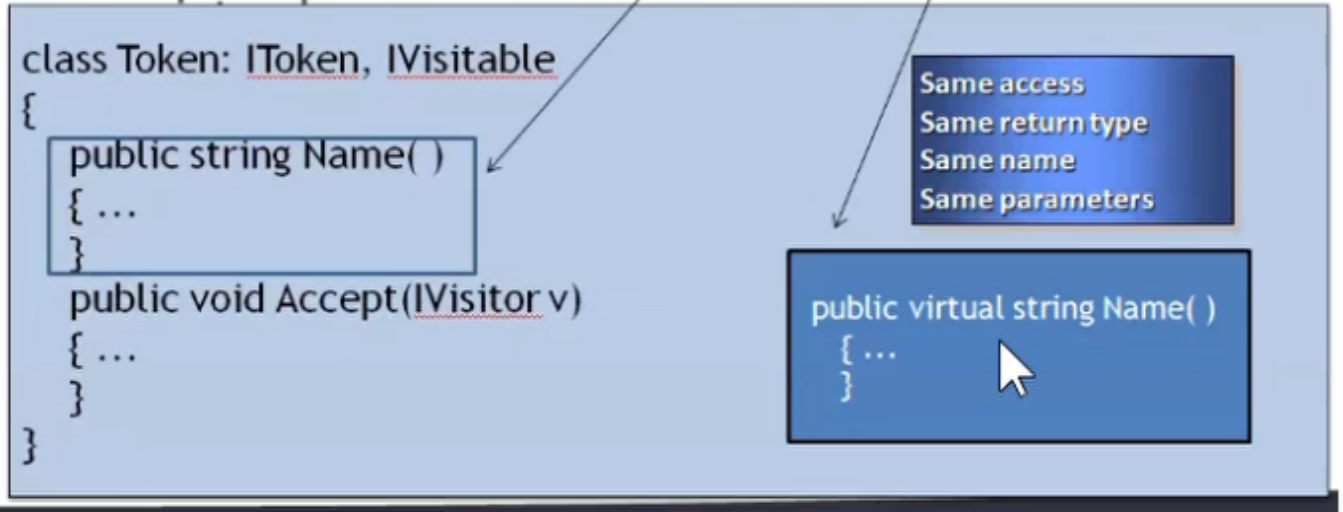
```
interface IToken
{
    string Name( );
}
interface IVisitable
{
    void Accept(IVisitor v);
}
class Token: IToken, IVisitable
{ ...
}
```



- An interface can extend zero or more interfaces
- A class can be more accessible than its base interfaces
- An interface cannot be more accessible than its base interfaces
- A class must implement all inherited interface methods

Implementing Interface Members Implicitly

- The implementing method must be the same as the interface method
- The implementing method can be non-virtual



Calling interface methods

- Calling interface methods Implicitly

```
Token tokenobject = new Token();
Tokenobject.Name();
```

Token class object created and method is being called through the object reference

```
Token tokenobject = new Token();
IToken itocken = tokenobject;
itocken.Name();
```

Token class object created and then reference is passed to a variable of the interface and through that reference variable method is being

Implementing Interface Methods Explicitly

- Use the fully qualified interface method name.
- No public keyword
- Restrictions of explicit interface method implementation
  - You can only access methods through the interface
  - You cannot declare methods as virtual
  - You cannot specify an access modifier

```
class Token: IToken, IVisitable
{
    string IToken.Name( )
    { ...
    }
    void IVisitable.Accept(IVisitor v)
    { ...
    }
}
```

```
class Program
{
Static void Main()
{
   Token tokenobj = new Token();
    IToken itoken = tokenobj;
    itoken.Name();
    Ivisitable ivisit = tokenobj;
    ivisit.Accept(ivisitor);
}
}
```

Calling interface methods

- Calling interface methods Implicitly

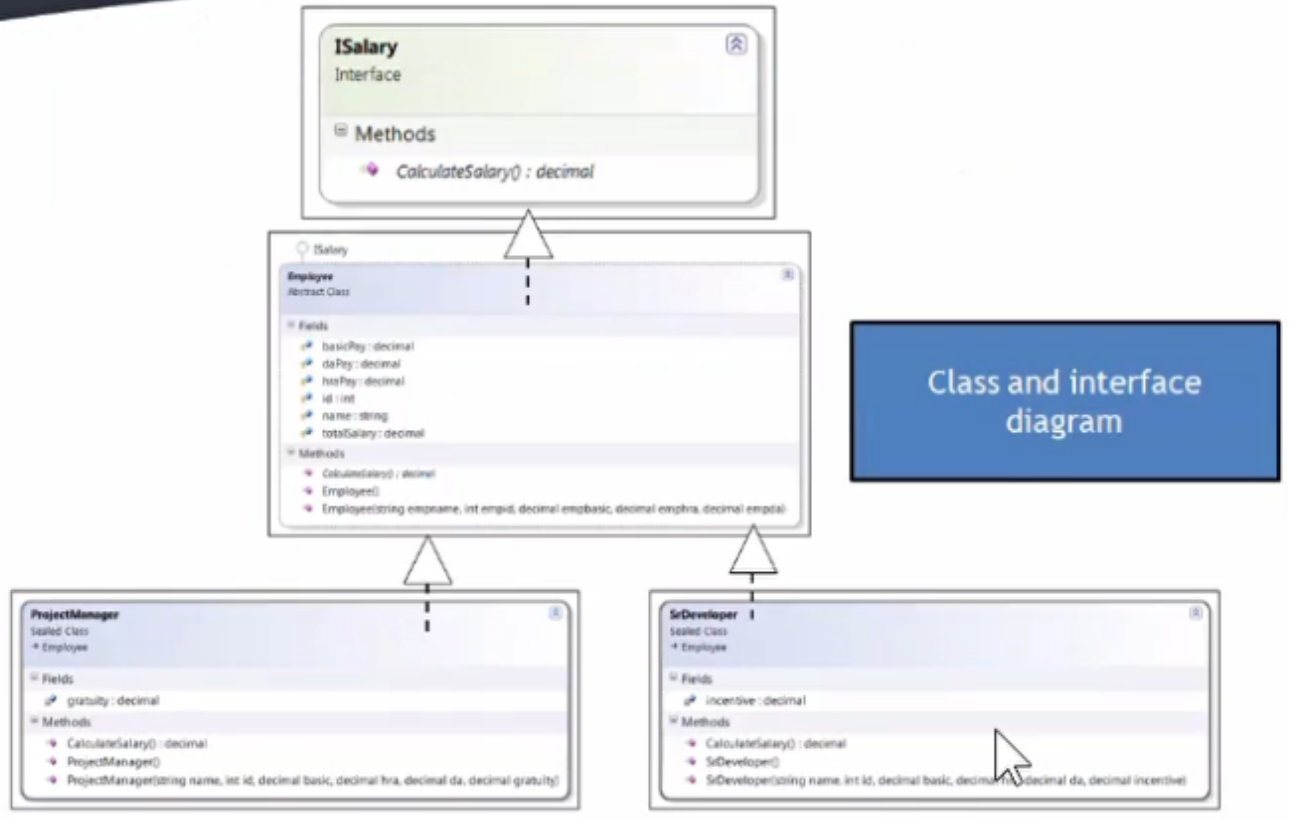
```
Token tokenobject = new Token();
Tokenobject.Name();
```

Token class object created and method is being called through the object reference

```
Token tokenobject = new Token();
IToken itocken = tokenobject;
itocken.Name();
```

Token class object created and then reference is passed to a variable of the interface and through that reference variable method is being

Example: Interface



Class and interface diagram

Example: Interface

ISalary interface and Employee class code

```
public interface ISalary
{
    decimal CalculateSalary();
}
```

```
abstract class Employee : ISalary
{
    protected string name;
    protected int id;
    protected decimal basicPay;
    protected decimal hraPay;
    protected decimal daPay;
    protected decimal totalSalary;

    public Employee() { }

    public Employee(string empname, int empid, decimal empbasic, decimal emphra, decimal empda)
    {
        this.name = empname;
        this.id = empid;
        this.hraPay = emphra;
        this.daPay = empda;
        this.basicPay = empbasic;
    }
    public abstract decimal CalculateSalary();
}
```

Example: Interface

Derived Class code

```
sealed class ProjectManager : Employee
{
    private decimal gratuity;

    public ProjectManager() { }

    public ProjectManager(string name, int id, decimal basic, decimal hra, decimal da, decimal gratuity)
        : base(name, id, basic, hra, da)
    {
        this.gratuity = gratuity;
    }

    public override decimal CalculateSalary()
    {
        this.totalSalary = (this.gratuity + this.basicPay + this.daPay + this.hraPay);
        return this.totalSalary;
    }
}
```

```
sealed class SrDeveloper : Employee
{
    private decimal incentive;

    public SrDeveloper() { }

    public SrDeveloper(string name, int id, decimal basic, decimal hra, decimal da, decimal incentive)
        : base(name, id, basic, hra, da)
    {
        this.incentive = incentive;
    }

    public override decimal CalculateSalary()
    {
        this.totalSalary = (this.incentive + this.basicPay + this.daPay + this.hraPay);
        return this.totalSalary;
    }
}
```

Comparing Abstract Classes to Interfaces

Abstract class	Interface
1. Can't be instantiated (similarity)	1. Can't be instantiated
2. Can't be sealed (similarity)	2. Can't be sealed
3. Contain abstract as well as non-abstract members	3. Can't contain any non-abstract members
4. Can contain fields (data members)	4. Can't contain fields (data members)
5. Can contain non-public members	Can't contain non-public members
6. By default members are private	6. By default members are public
7. Can extend interfaces (similarity)	7. Can extend interfaces
8. Can extend non-interfaces, such as another class	8. Can't extend non-interfaces

Structure

- A structure is a user-defined value type data type, which is a collection of different data type variables
- Memory space for structure variable, unlike class variable, is declared in the stack. This memory space is collective memory space of the containing variables
- It is used to encapsulate characteristics and behaviors of light-weight objects.
- Such as, Point can be represented as a struct rather than a class

Using a Structure

- Defining a Structure Type

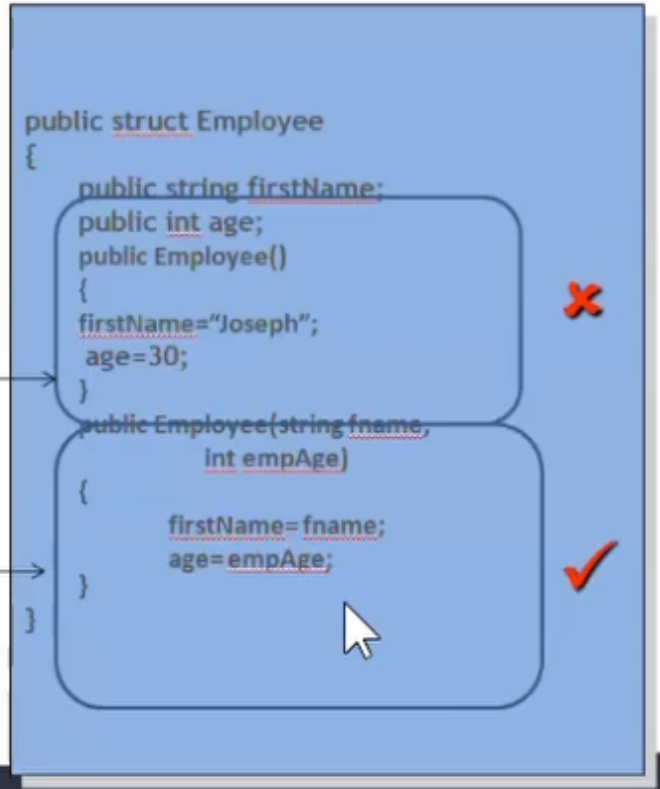
```
public struct Employee
{
    public string firstName;
    public int age;
}
```

- Declaring variable of a Structure Type

```
Employee companyEmployee;
companyEmployee.firstName = "Joe";
companyEmployee.age = 23;
--or—
Employee companyEmployee = new Employee();
//calling default constructor of structure to initialize fields of struct
companyEmployee.firstName = "Joe";
companyEmployee.age = 23;
```

Declaring a Constructor for a Struct

- The compiler
  - Always generates a default constructor. Default constructors automatically initialize all fields to zero.
- The programmer
  - Can never declare a default constructor.
  - Can never declare a protected constructor.
  - Can declare constructors with one or more arguments * (overloaded constructors). Declared constructors do not automatically initialize fields to zero.



Data Conversions

- The **is** Operator
- The **as** Operator
- Conversions and the object Type
- Boxing and Unboxing

The *is* Operator

- Returns true if a conversion can be made

```
Bird b;
if (a is Bird)
    b = (Bird) a; // Safe
else
    Console.WriteLine("Not a Bird");
```

The *as* Operator

- Converts between reference types, like cast
- On error
  - Returns null
  - Does not raise an exception

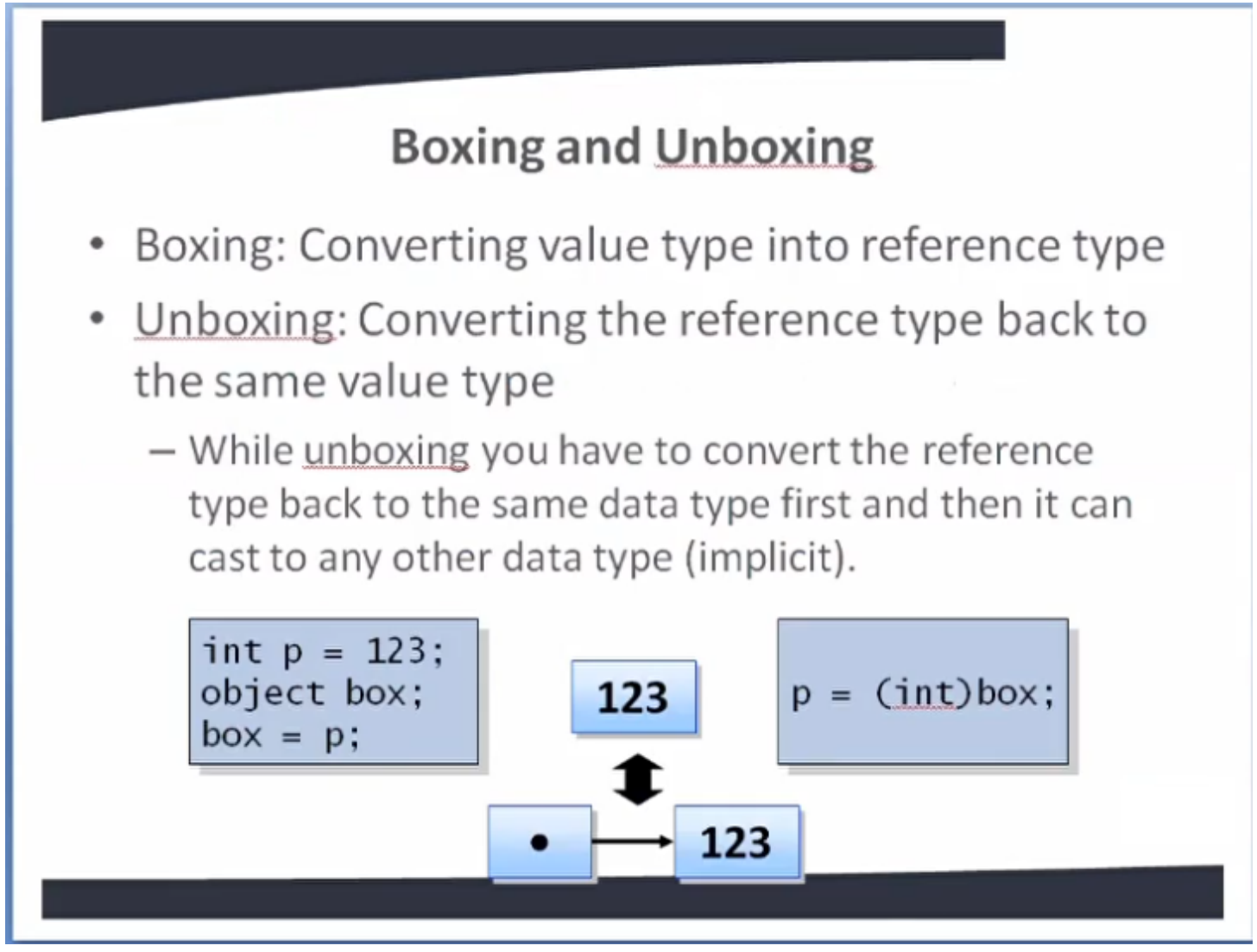
```
Bird b = a as Bird; // Convert

if (b == null)
      Console.WriteLine("Not a bird");
```

Conversions and the object Type

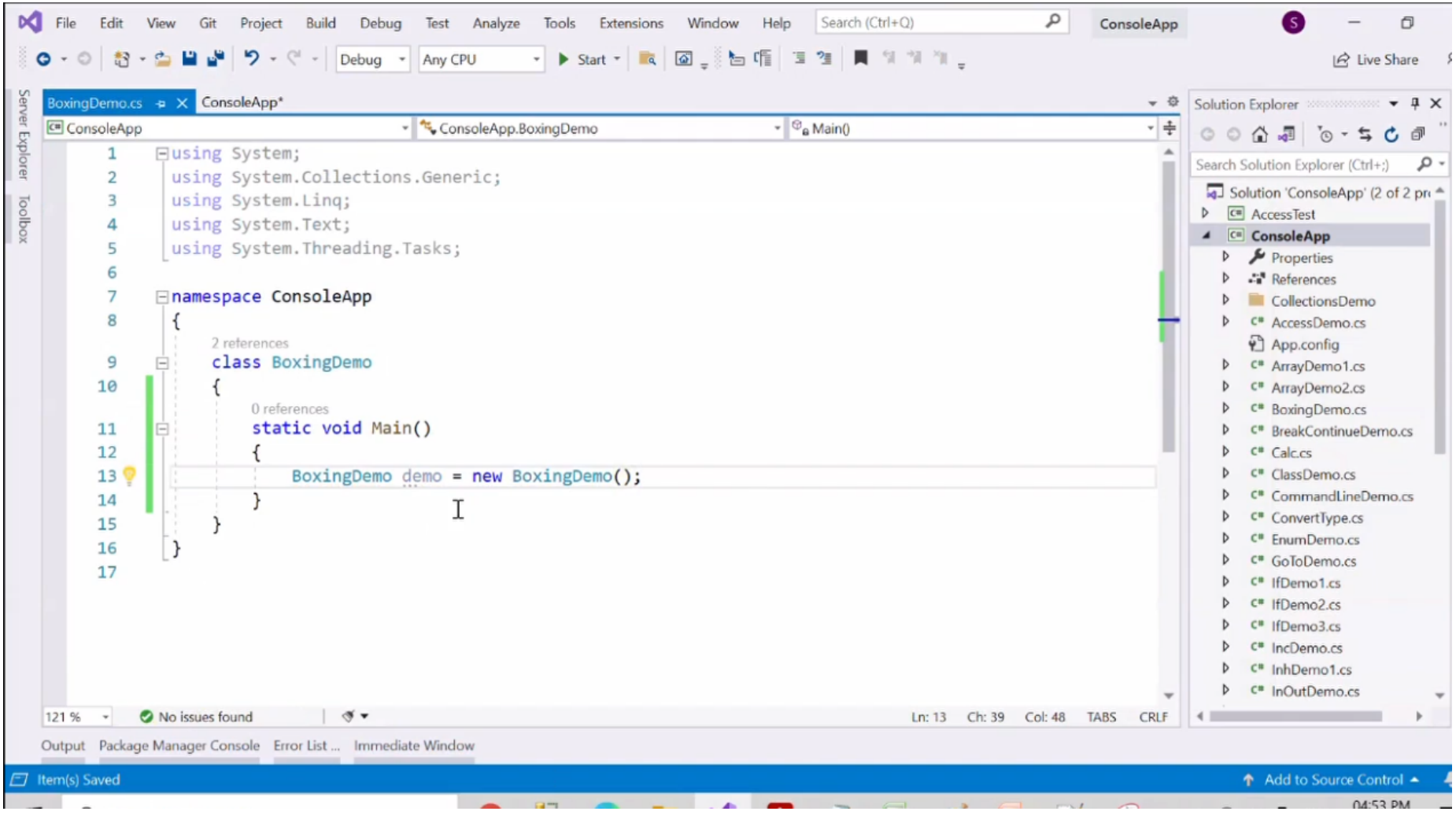
- The object type is the base for all classes
- Any reference can be assigned to object
- Any object variable can be assigned to any reference
  - With appropriate type conversion and checks
- The object type and is operator

```
object ox;
ox = a;
ox = (object) a;
ox = a as object;
```

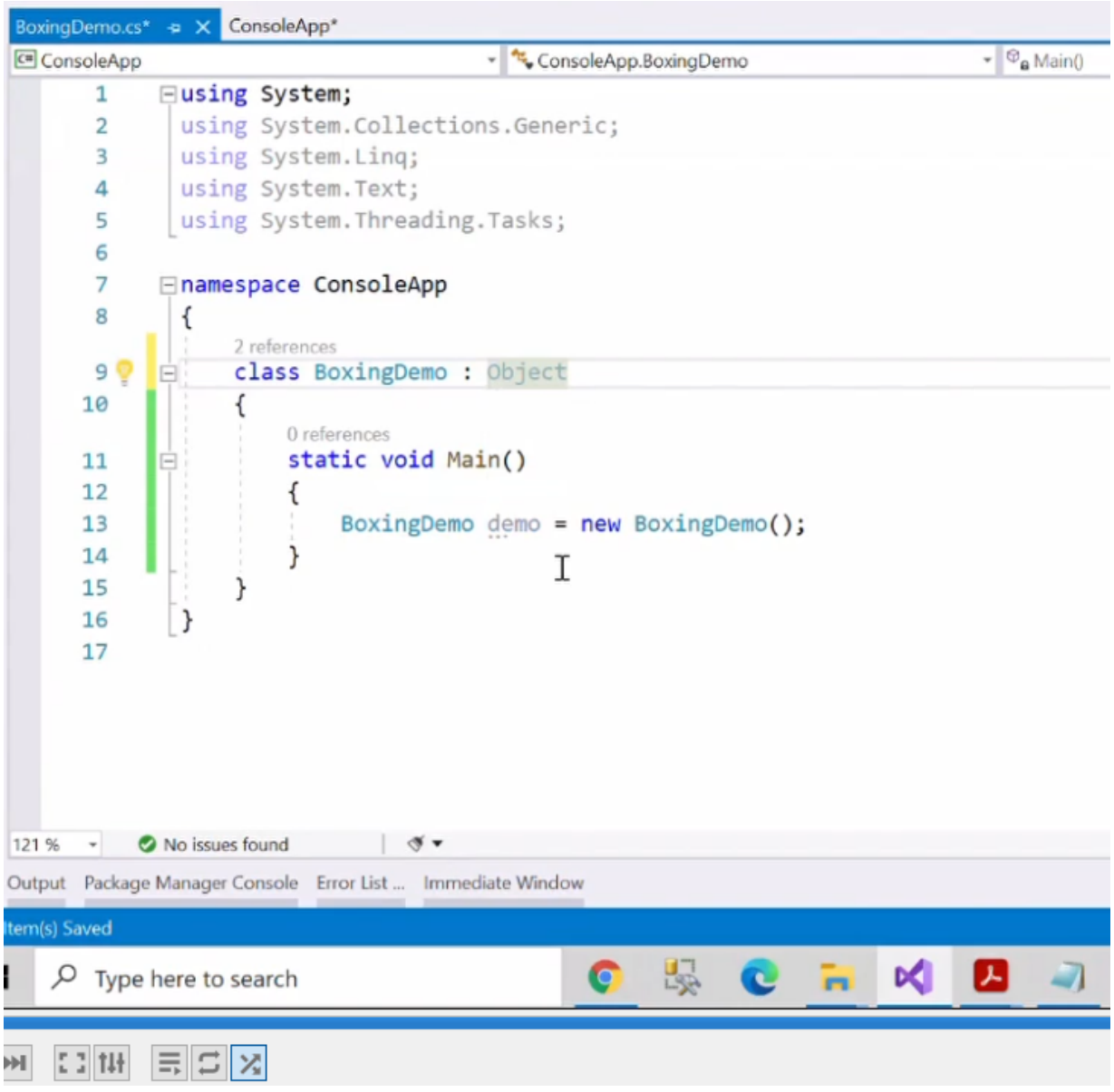
```
b = (Bird) ox;
b = ox as Bird;
```



What is the super class of this boxing class

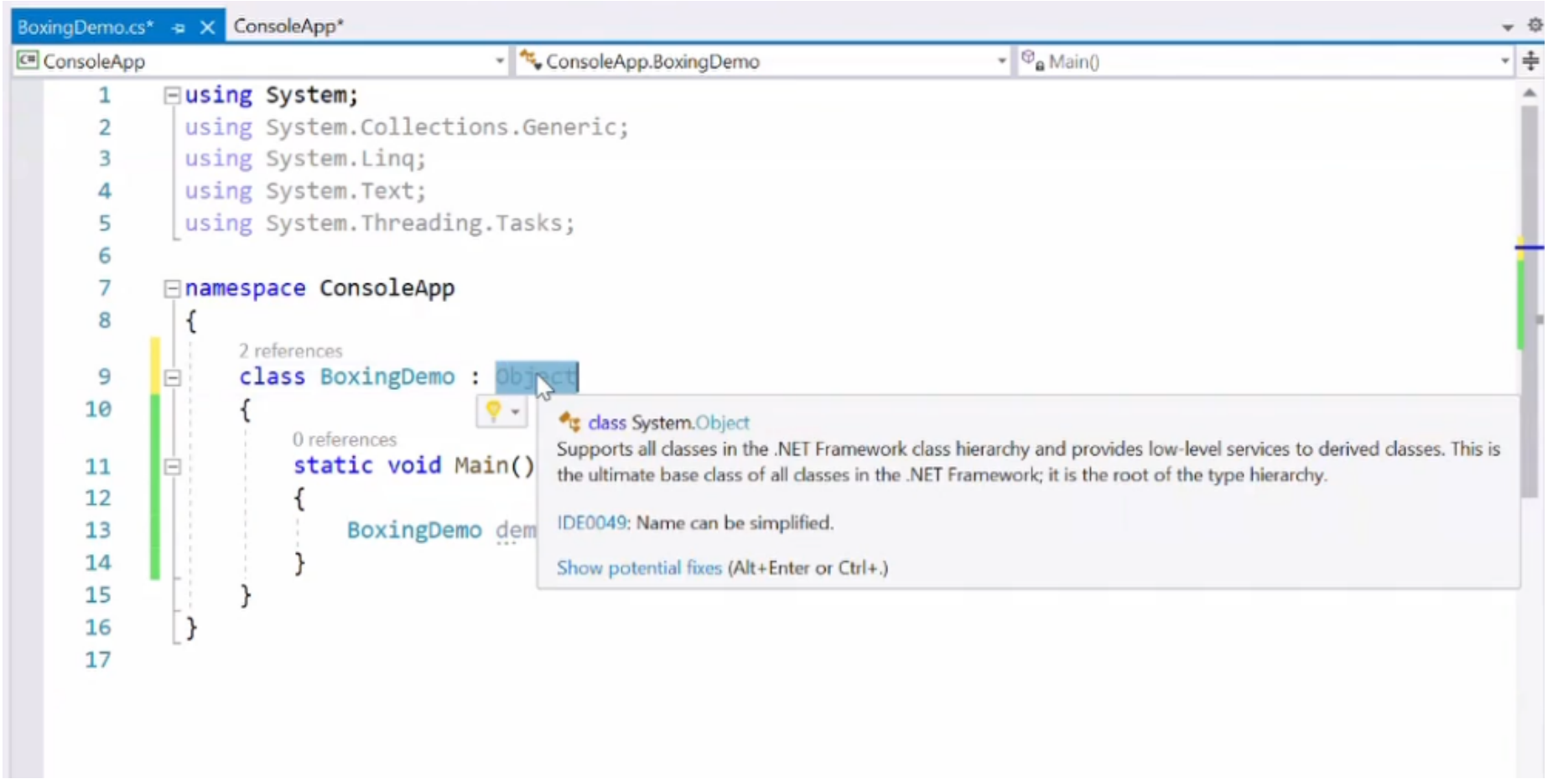


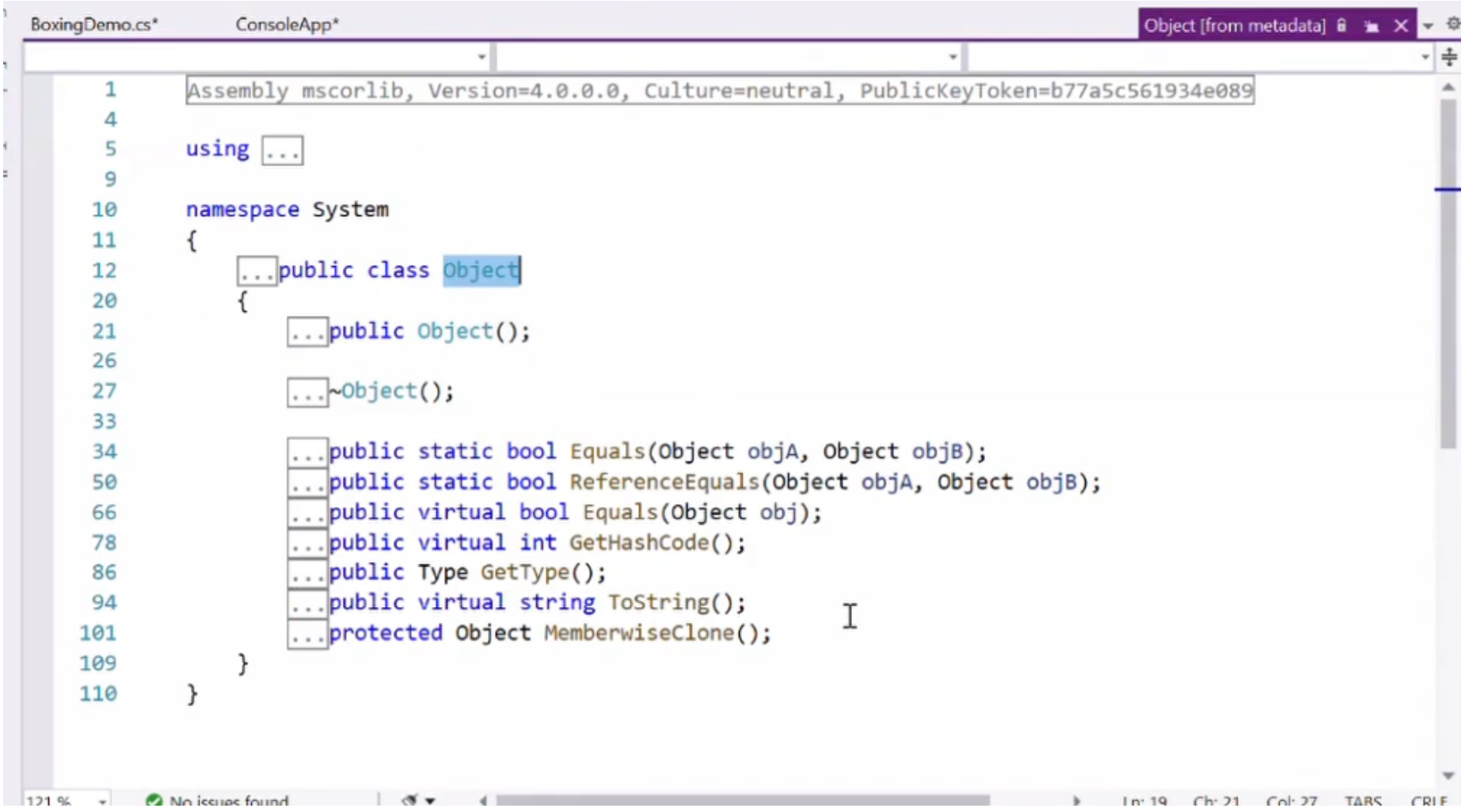
this is as simple as that me saying :Object ,So object becomes the main class


BoxingDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.BoxingDemo
Main()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
2 references
class BoxingDemo : Object
{
0 references
static void Main()
{
BoxingDemo demo = new BoxingDemo();
}
}
}
121 %
No issues found
Output
Package Manager Console
Error List ...
Immediate Window
Item(s) Saved
Type here to search

So Object is the super classes of all this classes

What is there in my Object class


BoxingDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.BoxingDemo
Main()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
2 references
class BoxingDemo : Object
{
0 references
static void Main()
{
BoxingDemo dem
}
}
}
class System.Object
Supports all classes in the .NET Framework class hierarchy and provides low-level services to derived classes. This is the ultimate base class of all classes in the .NET Framework; it is the root of the type hierarchy.
IDE0049: Name can be simplified.
Show potential fixes (Alt+Enter or Ctrl+.)



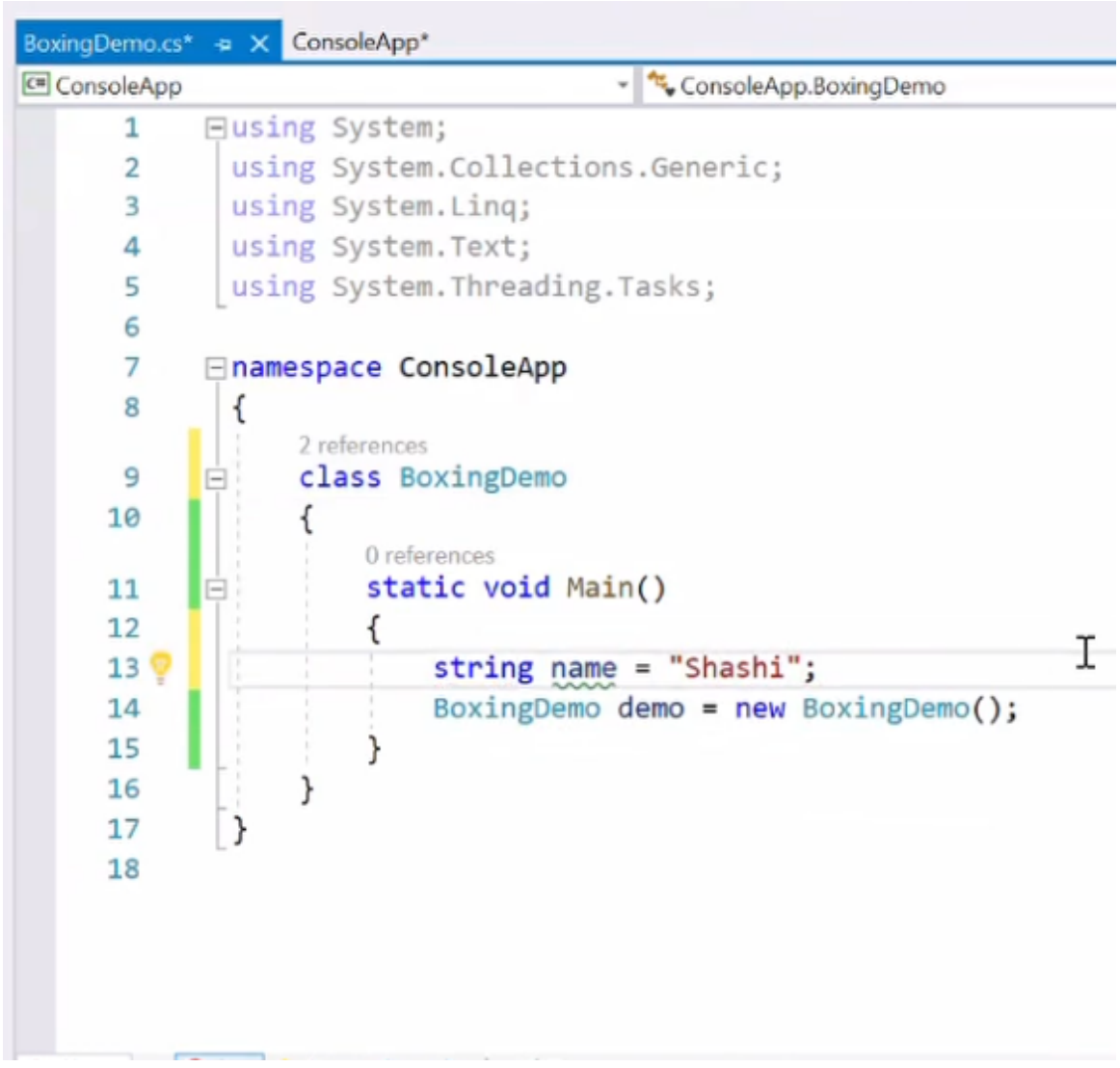
So my object class has a object constructor which is not taking any parameter

at line 27 you can see a tilt~ symbol here this are called

Distructors

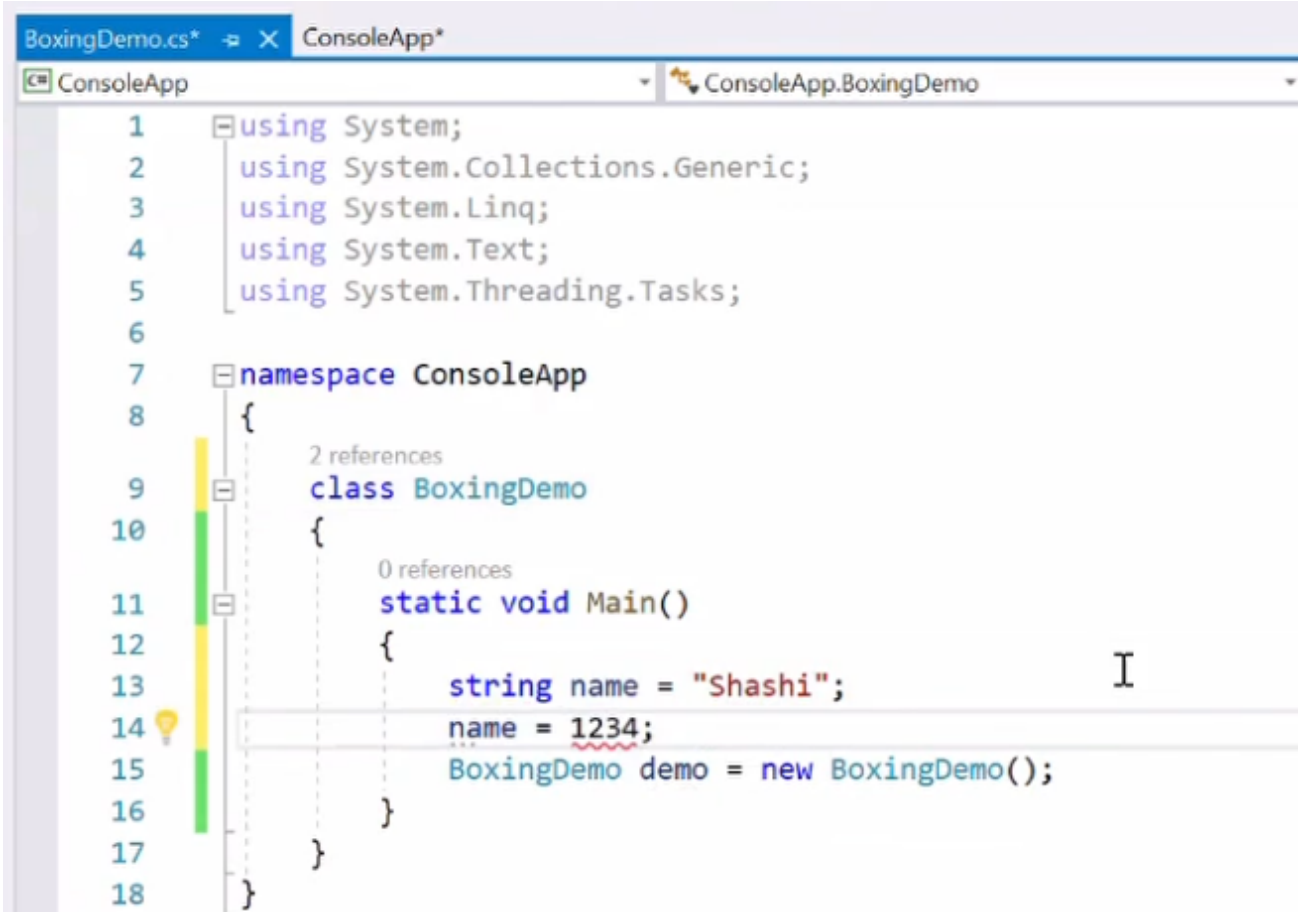
As we know constructor use to construct
in the same way Distructor is use to Distroy
What is that I can destroy here -------- In Constructor we are gone to allocate the memory, In Destructor we can deallocate the memory
all those details will be written the Distructor
and object will also has the Equals method and GetHashcode we'll see that in some time
it also has some other methods

So Object Class as it as base class It is a GOD
me, my father, my grand father ..... if we keep moving up the ladder So there would be some one who have created all of us and we call that energy as GOD
So Object is like god to this program, C# or java programming beyond object we don't have anybody, so object is a base class
Now when i want to store any data

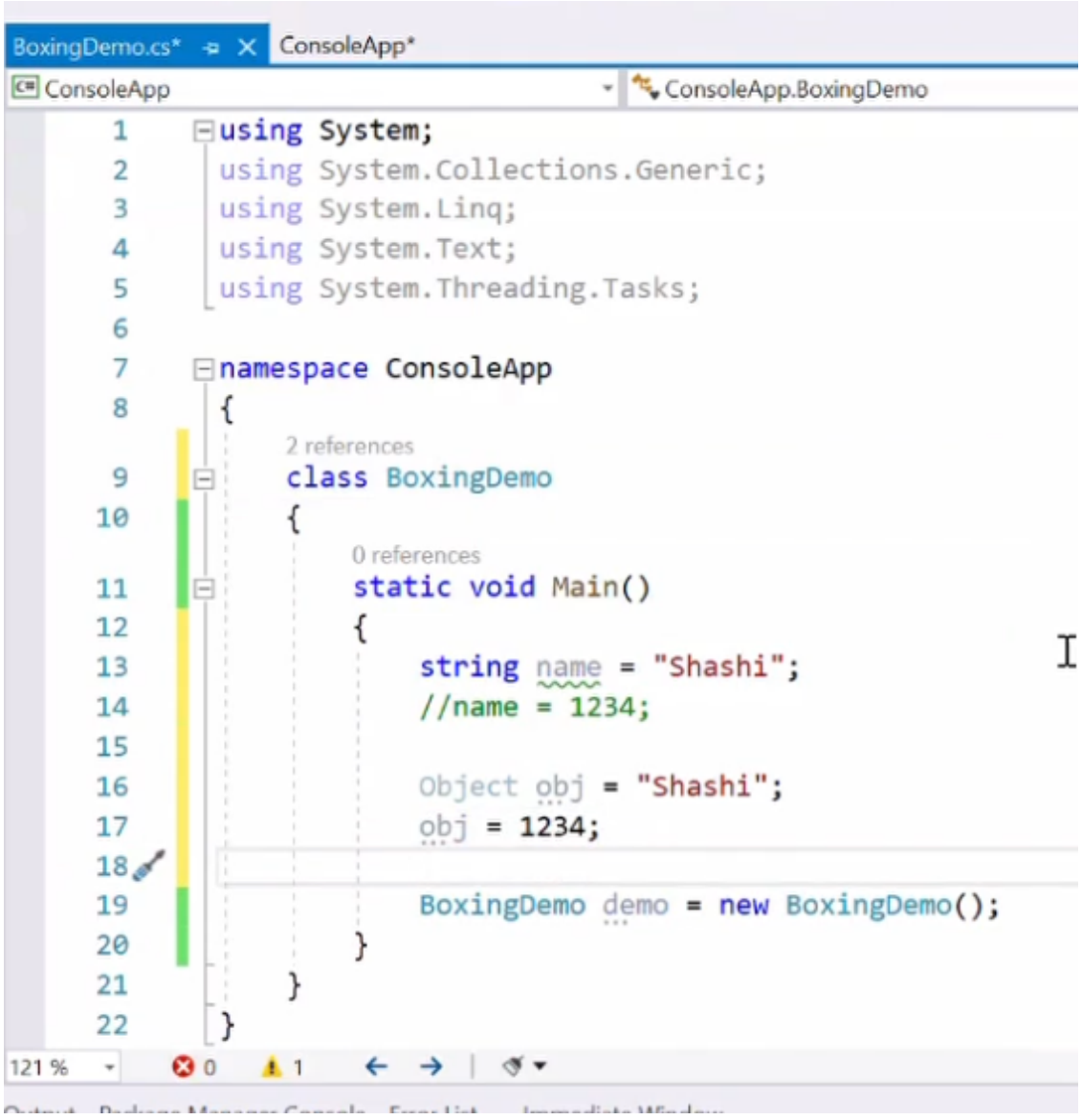


when i say string here, so I can store only string data here right, I cannot store anything else here right

can I say name = 1234 ------No it is not possible



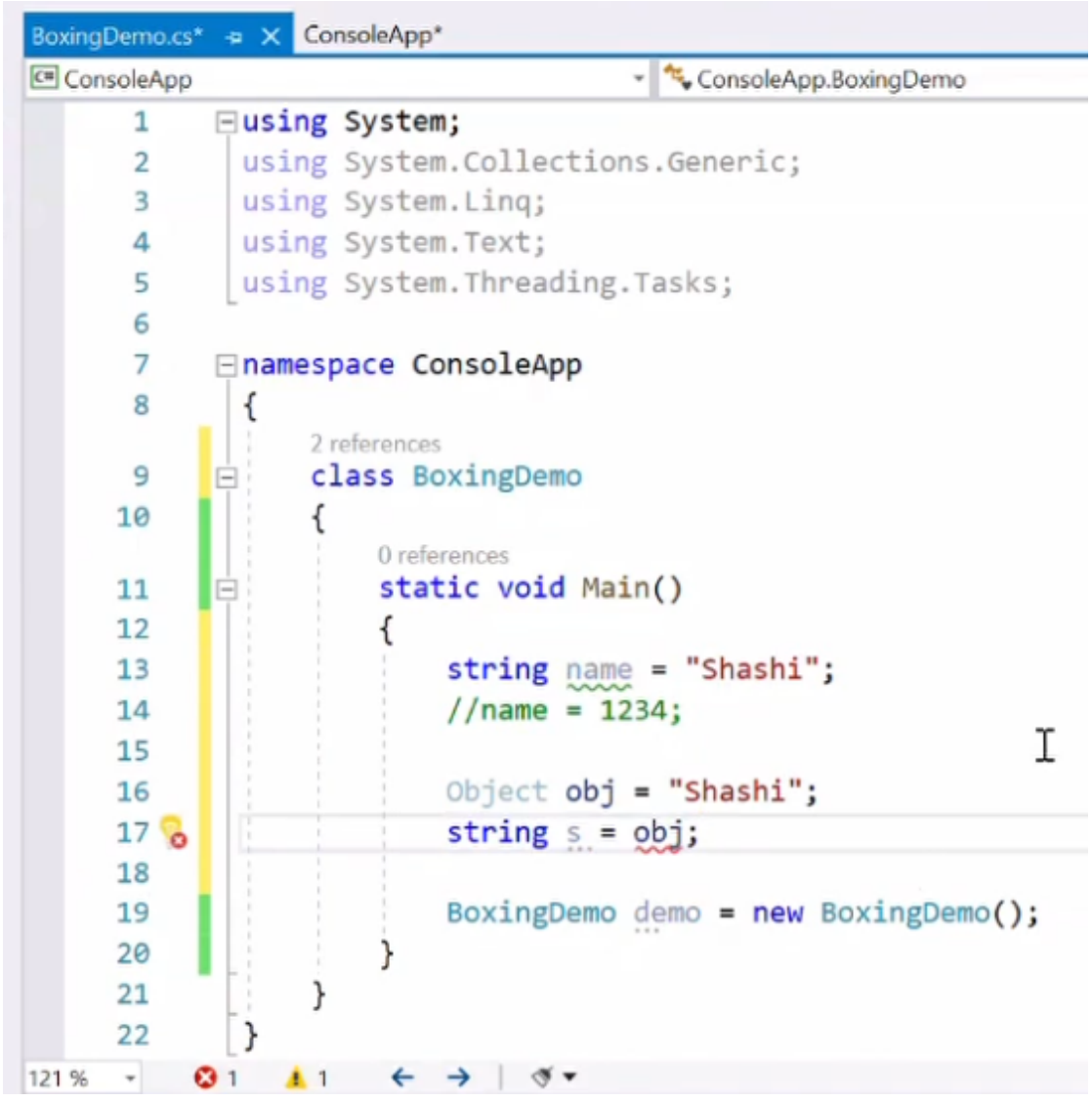
but where as with object it is not the case, It will never complaint


BoxingDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.BoxingDemo
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
2 references
class BoxingDemo
{
0 references
static void Main()
{
string name = "Shashi";
//name = 1234;
Object obj = "Shashi";
obj = 1234;
BoxingDemo demo = new BoxingDemo();
}
}
}
121 %
0
1

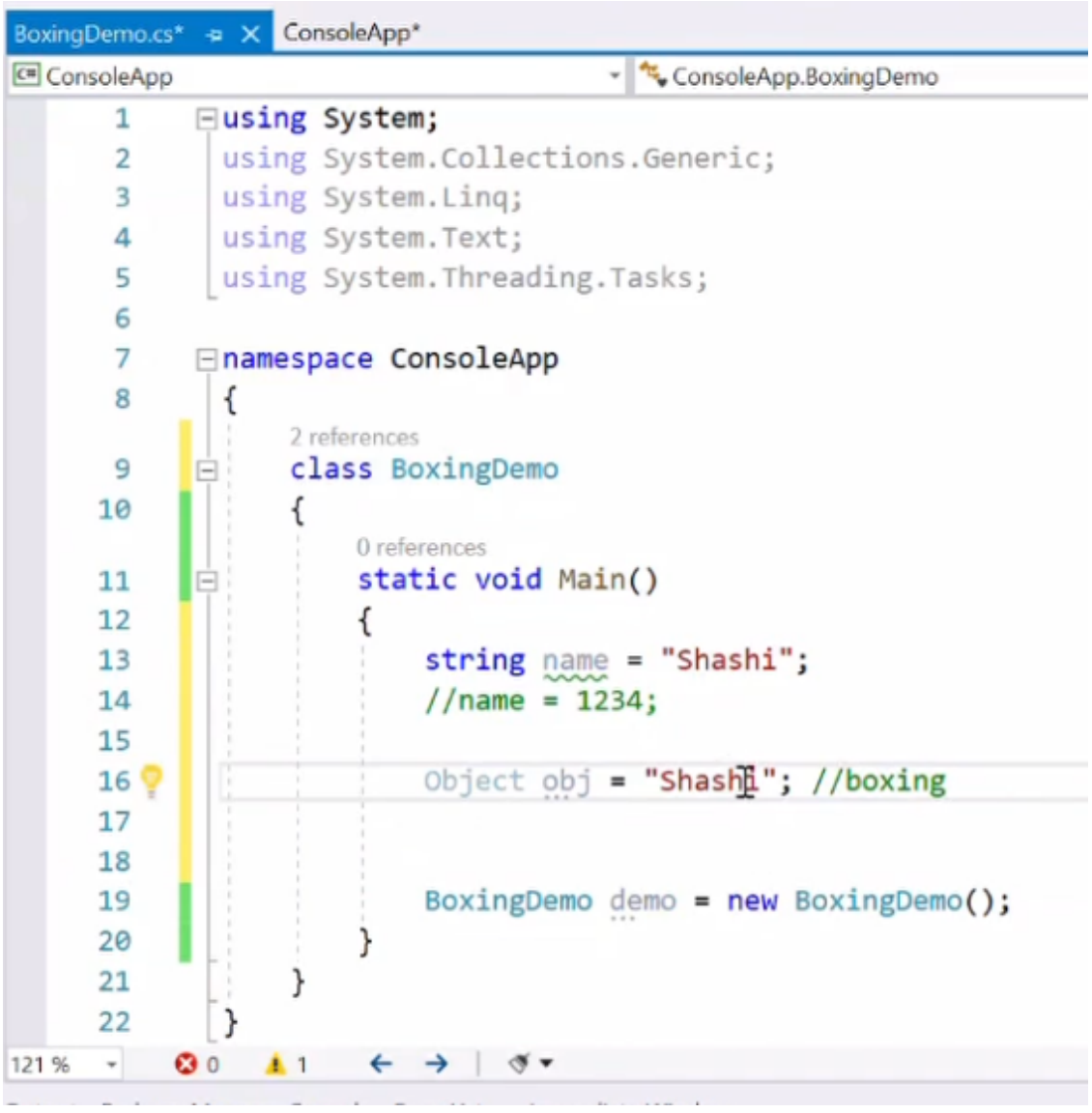
meaning inside object I can push anything
you can push employee
you can push integer

So Object will Encapsulate Everything

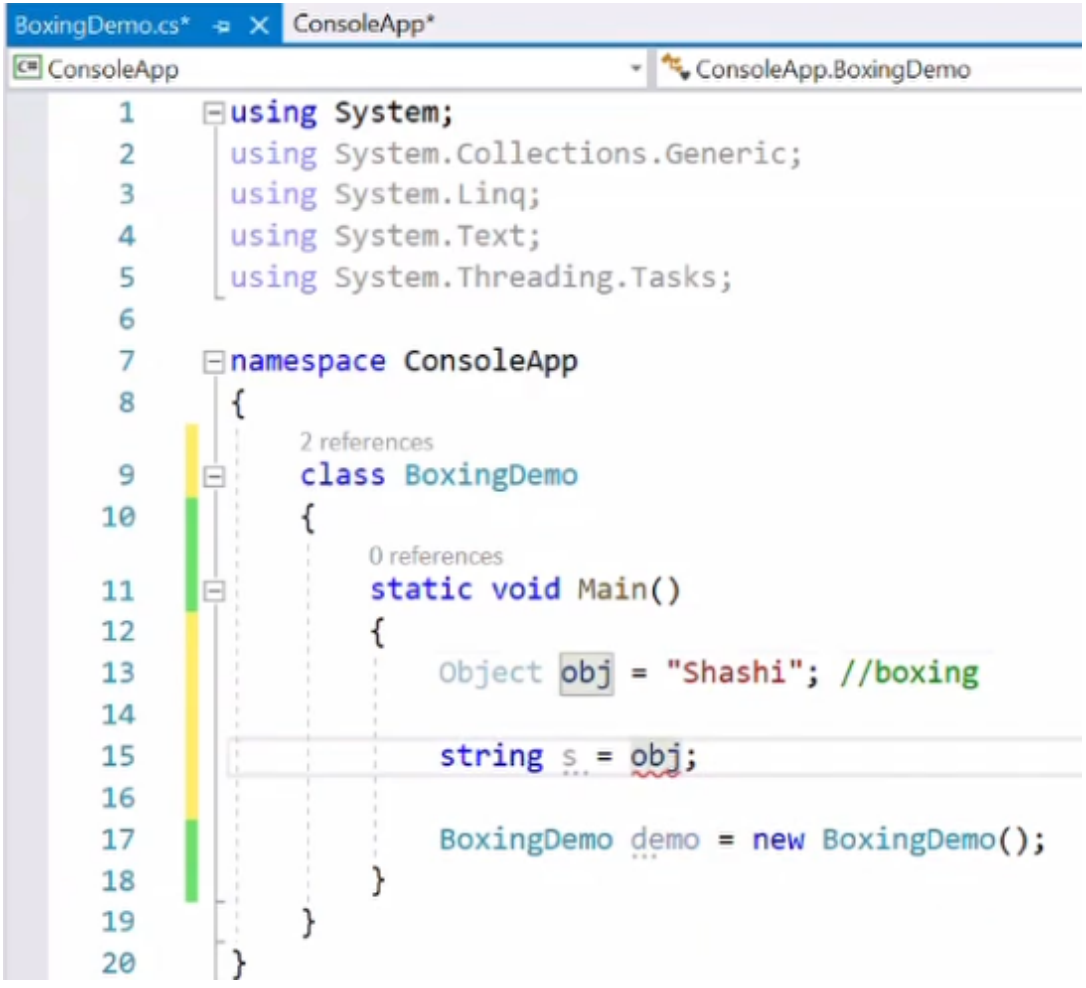
Now if object have taken this string , Can I directly print this value
--------- this is not possible


BoxingDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.BoxingDemo
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
2 references
class BoxingDemo
{
0 references
static void Main()
{
string name = "Shashi";
//name = 1234;
Object obj = "Shashi";
string s = obj;
BoxingDemo demo = new BoxingDemo();
}
}
}
121 %
1
1

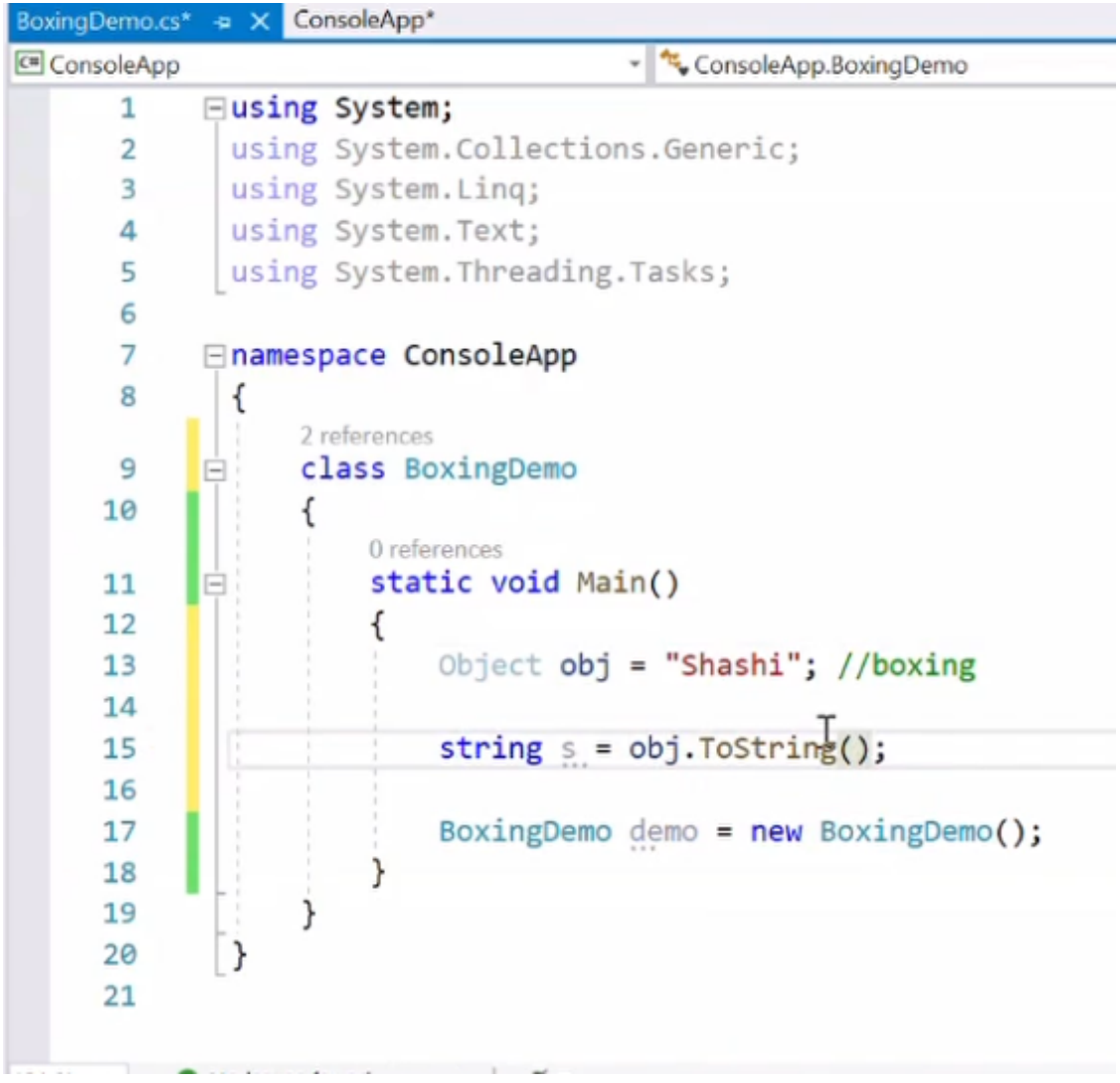
So i'm trying to store a general value in the object - This is called **Boxing** - from the Primitive type it is trying to store the Object type



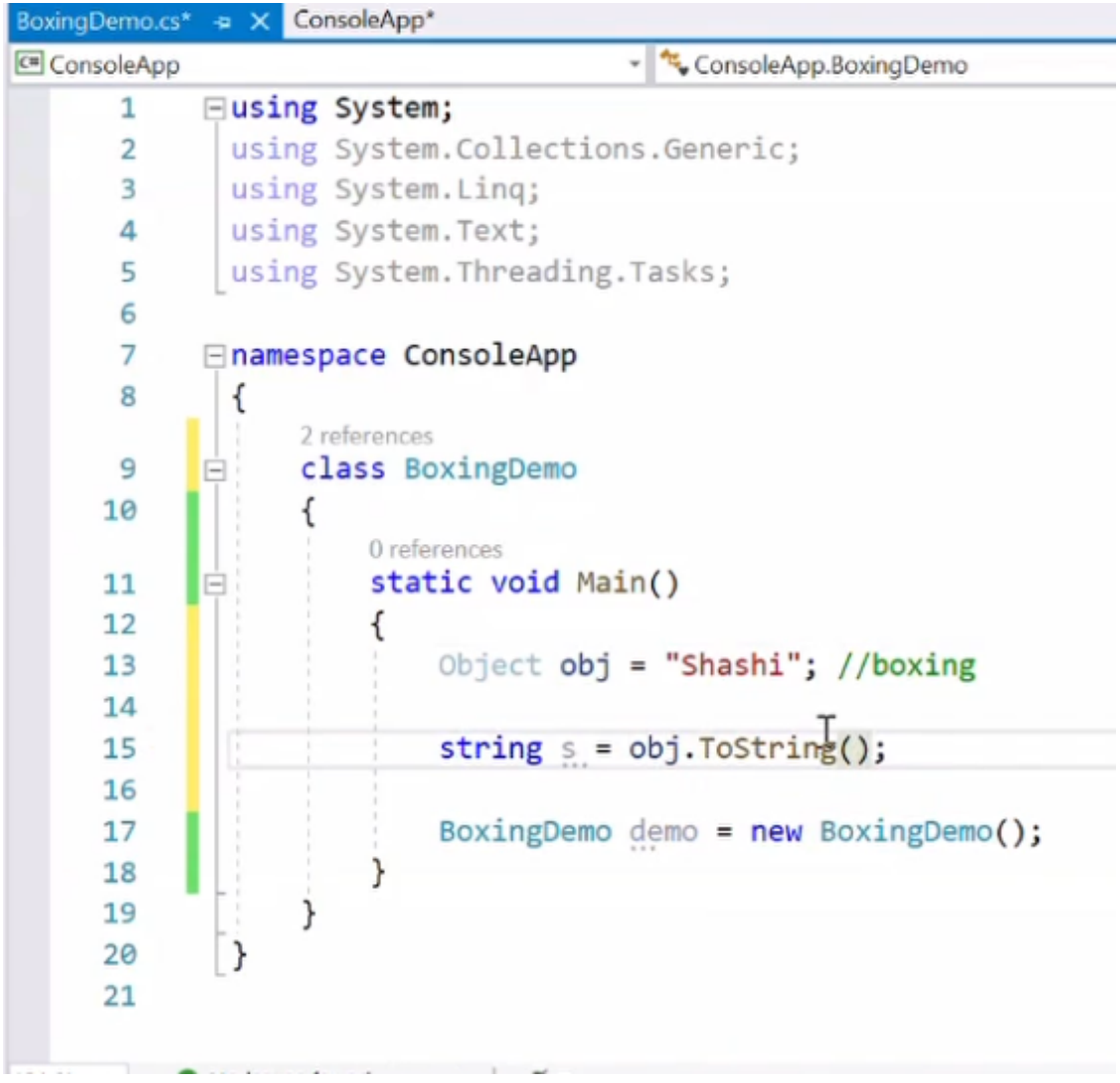
Now when i say string s = obj, it is not possible



So you need to use predefined method .Tostring()


BoxingDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.BoxingDemo
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
2 references
class BoxingDemo
{
0 references
static void Main()
{
Object obj = "Shashi"; //boxing
string s = obj.ToString();
BoxingDemo demo = new BoxingDemo();
}
}
}

So this Is Called UnBoxing - So What is been wrapped inside in the boxing I'm trying to take that particular thing out and I'm pushing it back


BoxingDemo.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.BoxingDemo
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
2 references
class BoxingDemo
{
0 references
static void Main()
{
Object obj = "Shashi"; //boxing
string s = obj.ToString();
BoxingDemo demo = new BoxingDemo();
}
}
}

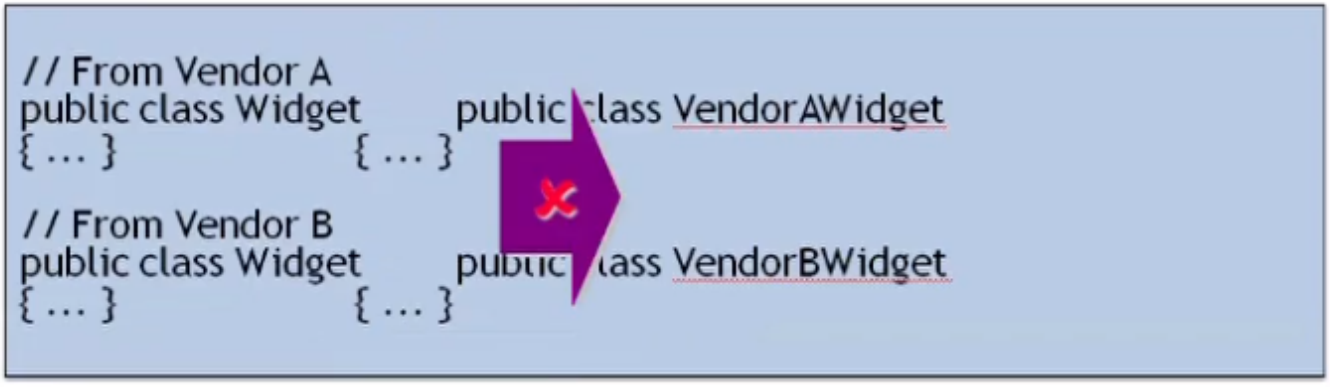
Scope

- The scope of a name is the region of program text in which you can refer to the name without qualification

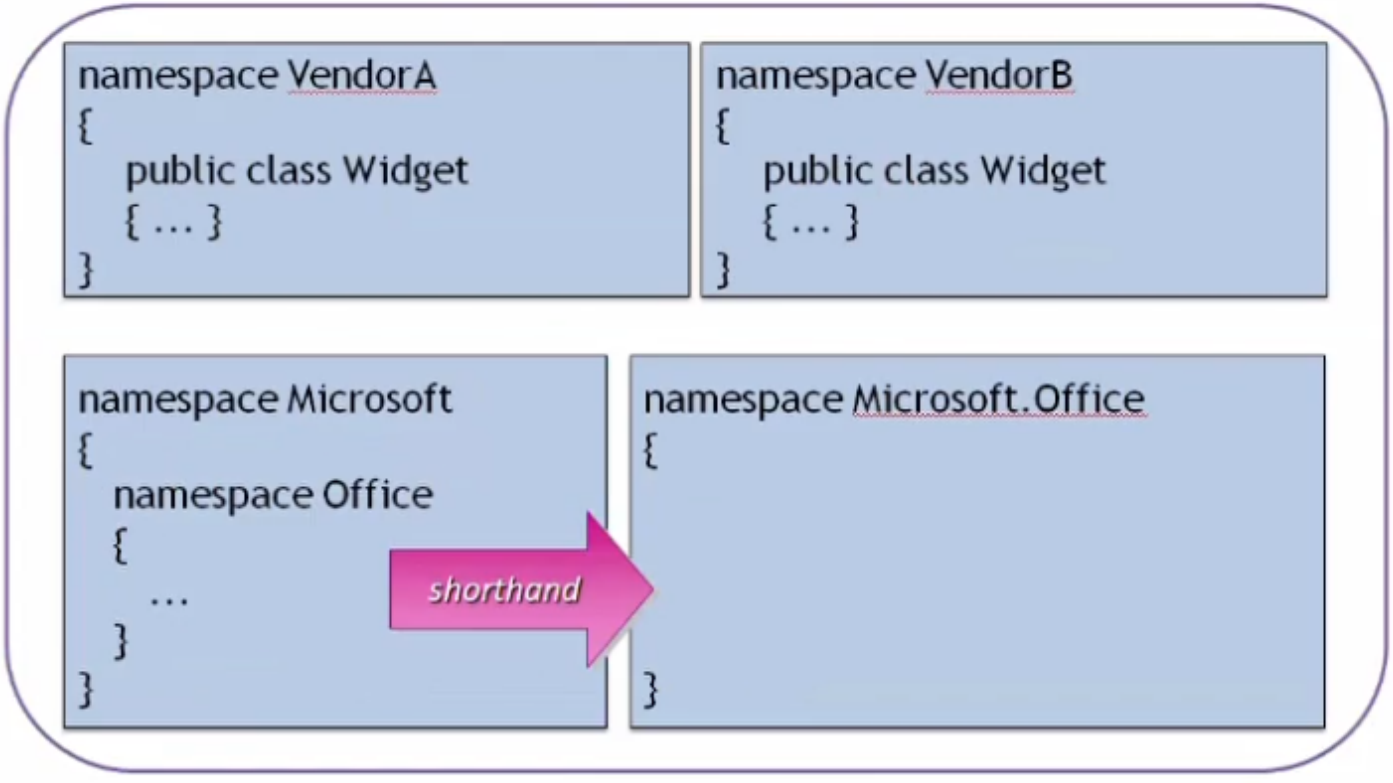
```
public class Bank
{
   public class Account
   {
      public void Deposit(decimal amount)
      {
         balance += amount;
      }
      private decimal balance;
   }
   public Account OpenAccount( ) { ... }
}
```

Resolving Name Clashes

- Consider a large project that uses thousands of classes
- What if two classes have the same name?
- Do not add prefixes to all class names



Declaring Namespaces



Fully Qualified Names

- A fully qualified class name includes its namespace
- Unqualified class names can only be used in scope

```
namespace VendorA
{
    public class Widget { … }
    …
}
class Application
{
    static void Main( )
    {
        Widget w = new Widget( );
        VendorA.Widget w = new VendorA.Widget( );
    }
}
```

Declaring using-alias-directives

- Creates an alias for a deeply nested namespace or type

```
namespace VendorA.SuiteB
{
    public class Widget { ... }
}
```

```
using Widget = VendorA.SuiteB.Widget;

class Application
{
    static void Main( )
    {
        Widget w = new Widget( );
    }
}
```

Guidelines for Naming Namespaces

- Use PascalCasing to separate logical components
  - Example: VendorA.SuiteB
- Prefix namespace names with a company name or well-established brand
  - Example: Microsoft.Office
- Use plural names when appropriate
  - Example: System.Collections
- Avoid name clashes between namespaces and classes

Quiz

Never let your curiosity die!

Can You Answer These Questions?

- What is the output of the following program?

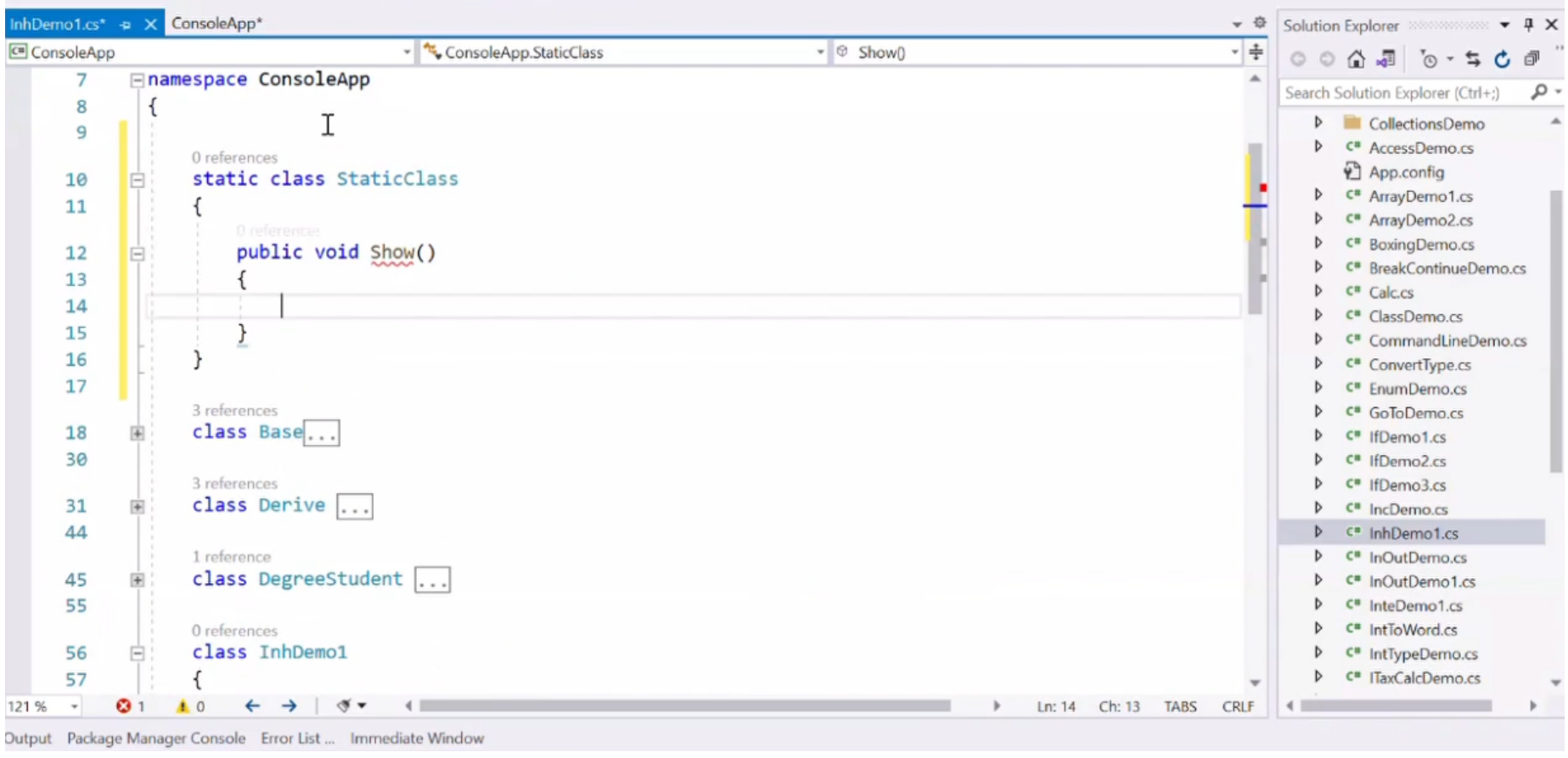
```
class Circle
{
        double radius;
        static Circle(double radius)
        {
                this.radius = radius;
        }
        public double getRadius()
        {
                return radius;
        }
}
public class Tester
{
        public static void main(String[] args)
        {
                Circle circle = new Circle();
                Console.WriteLine(circle.getRadius());
        }
}
```

Now what is Sealed Classes ?

Sealed Classes - are those classes which will help me to

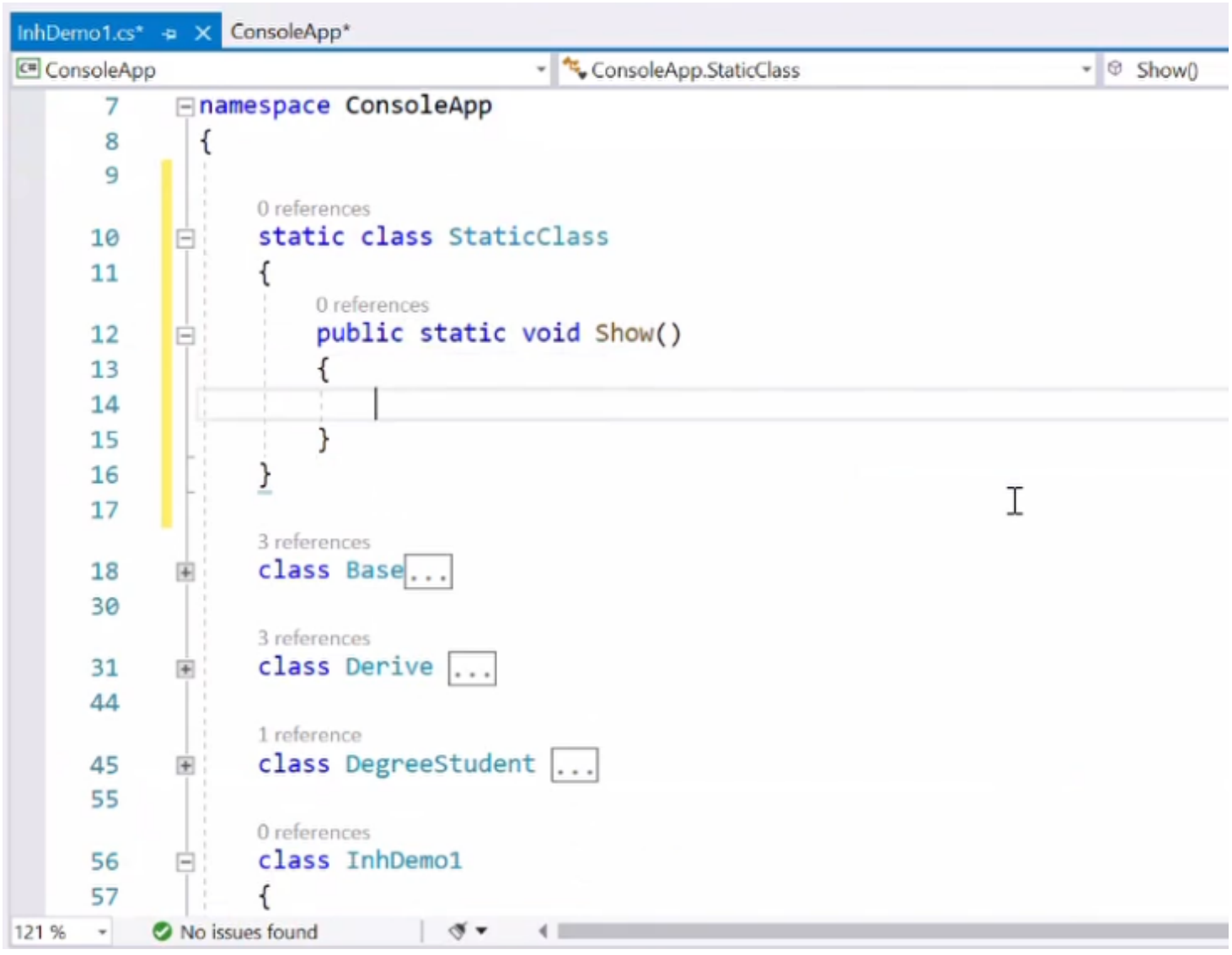
you have a base class and a derived class

Now Imagine I'll create a static class and tried to put public method inside it



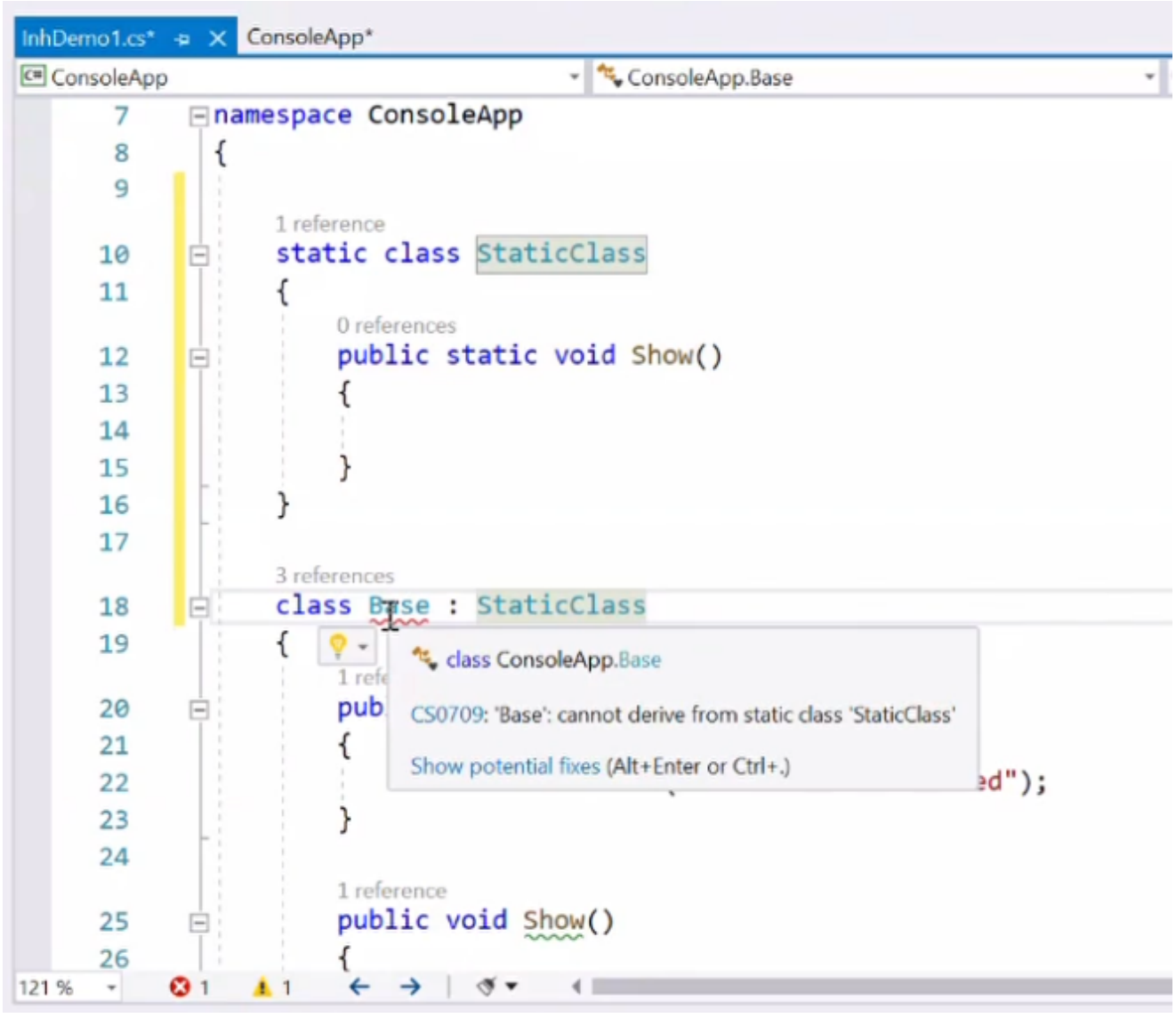
look at this it gaves me error

it says boss the show method what you are declaring here is not static, meaning any method inside the static class should always be static

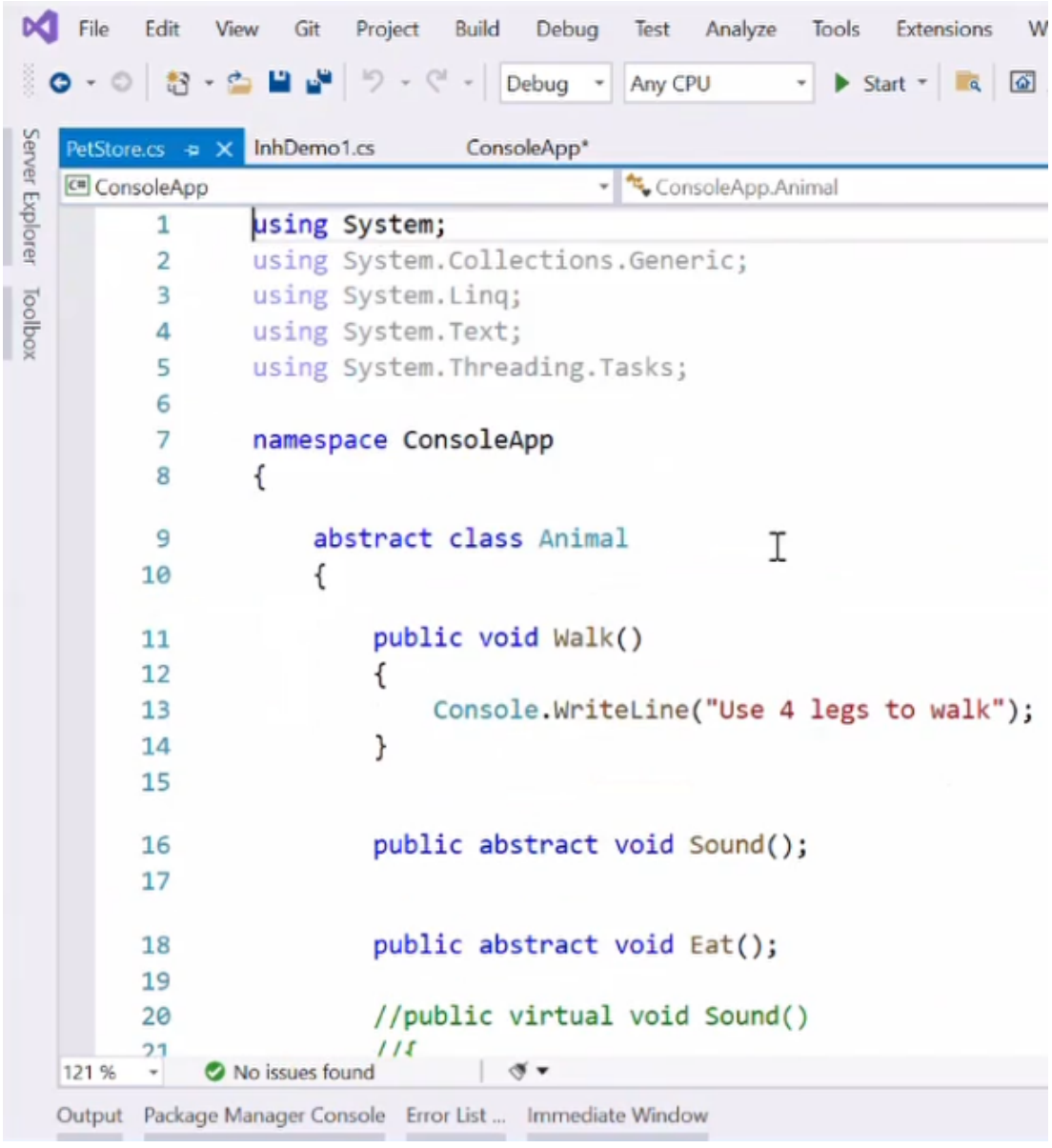

InhDemo1.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.StaticClass
Show()
namespace ConsoleApp
{
0 references
static class StaticClass
{
0 references
public static void Show()
{
}
}
3 references
class Base...
3 references
class Derive ...
1 reference
class DegreeStudent ...
0 references
class InhDemo1
{
121 %
No issues found

Now can you inherit static class

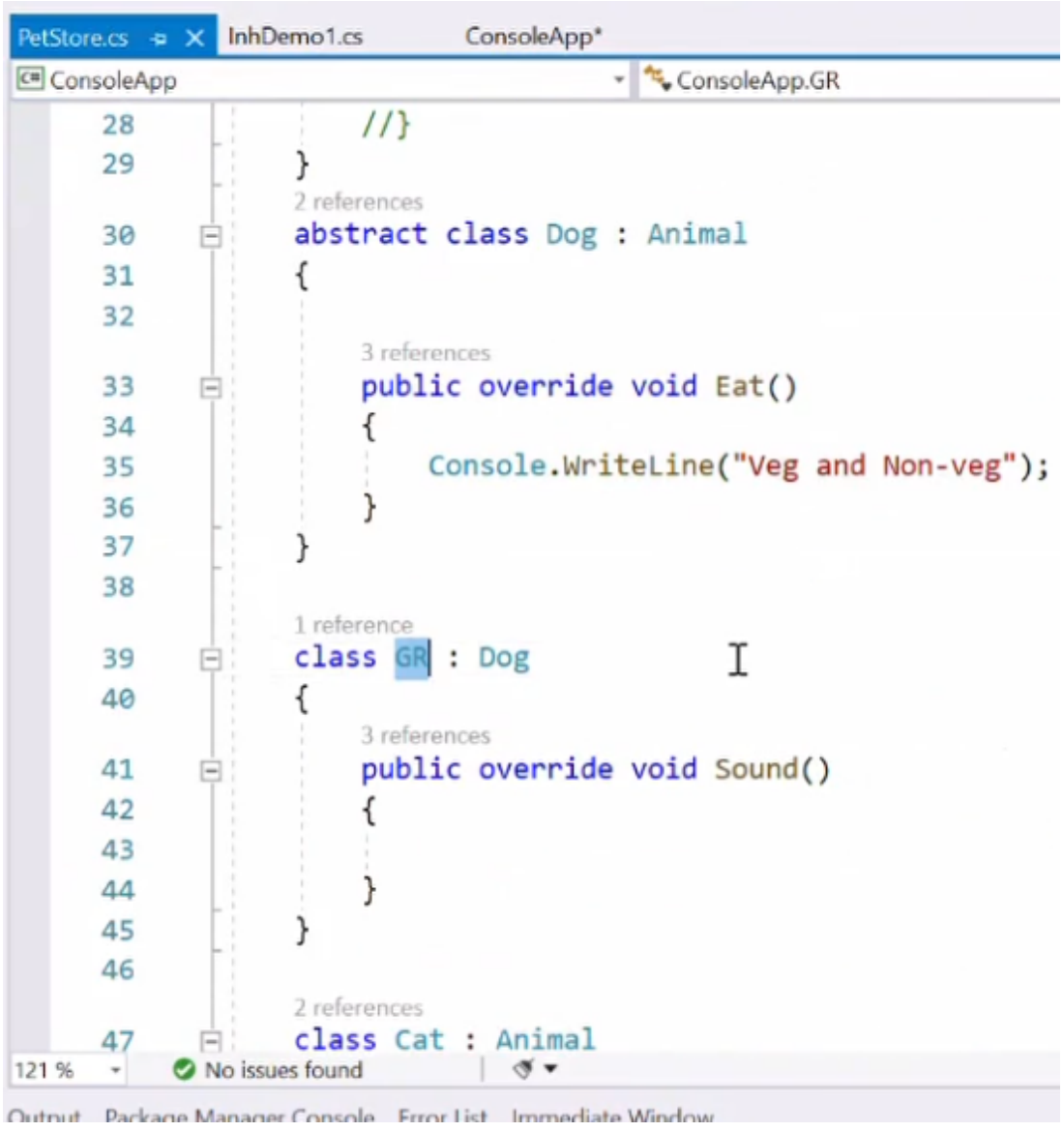
----- No


InhDemo1.cs*
ConsoleApp*
ConsoleApp
ConsoleApp.Base
namespace ConsoleApp
{
1 reference
static class StaticClass
{
0 references
public static void Show()
{
}
}
3 references
class Base : StaticClass
{
class ConsoleApp.Base
CS0709: 'Base': cannot derive from static class 'StaticClass'
Show potential fixes (Alt+Enter or Ctrl+.)
1 reference
public void Show()
{
121 %
1
1

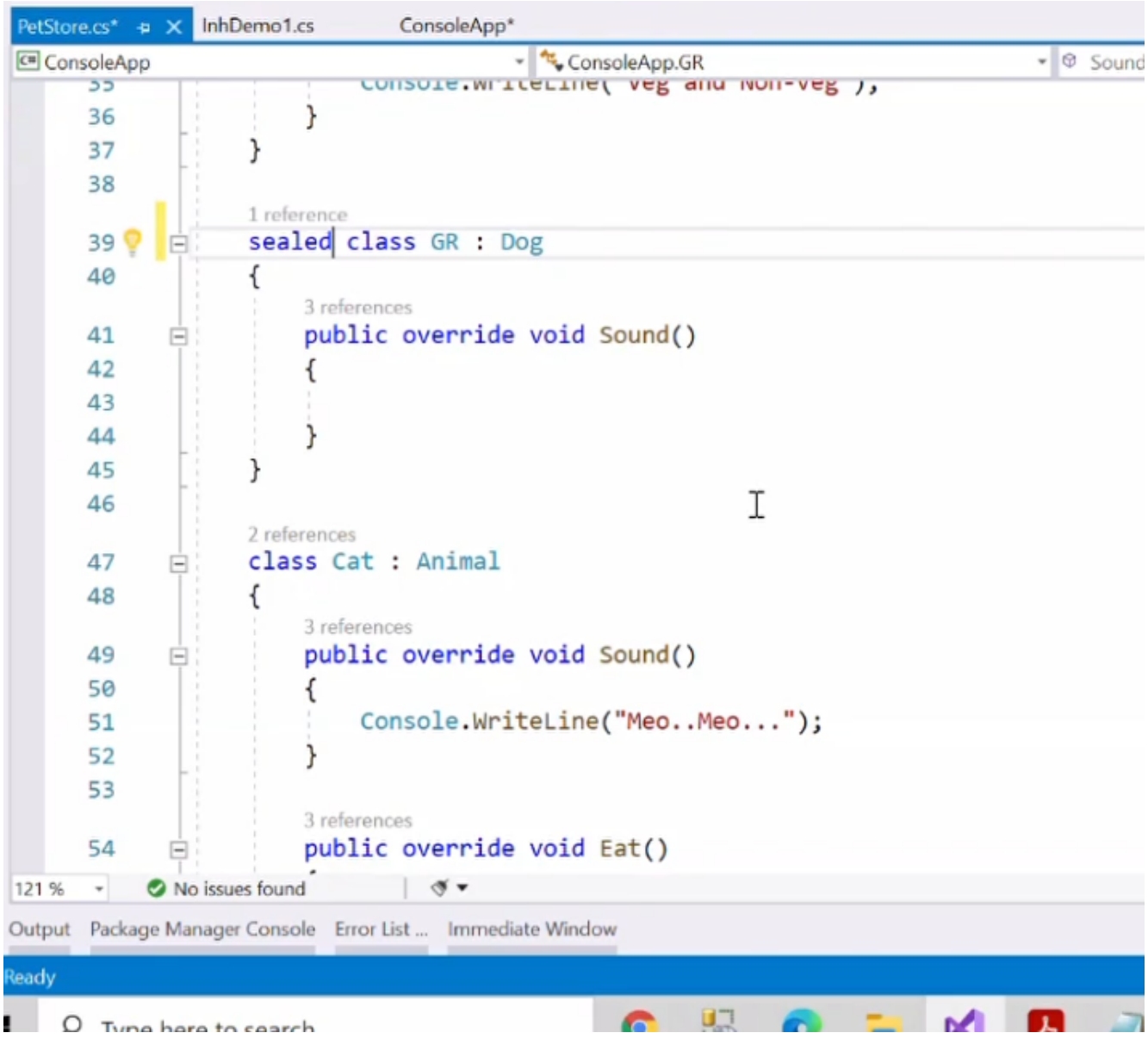
No In Petstore we have Animal class as Abstract


File Edit View Git Project Build Debug Test Analyze Tools Extensions W
Debug
Any CPU
Start
Server Explorer
Toolbox
PetStore.cs
InhDemo1.cs
ConsoleApp*
ConsoleApp
ConsoleApp.Animal
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
abstract class Animal
{
public void Walk()
{
Console.WriteLine("Use 4 legs to walk");
}
public abstract void Sound();
public abstract void Eat();
//public virtual void Sound()
121 %
No issues found
Output
Package Manager Console
Error List ...
Immediate Window

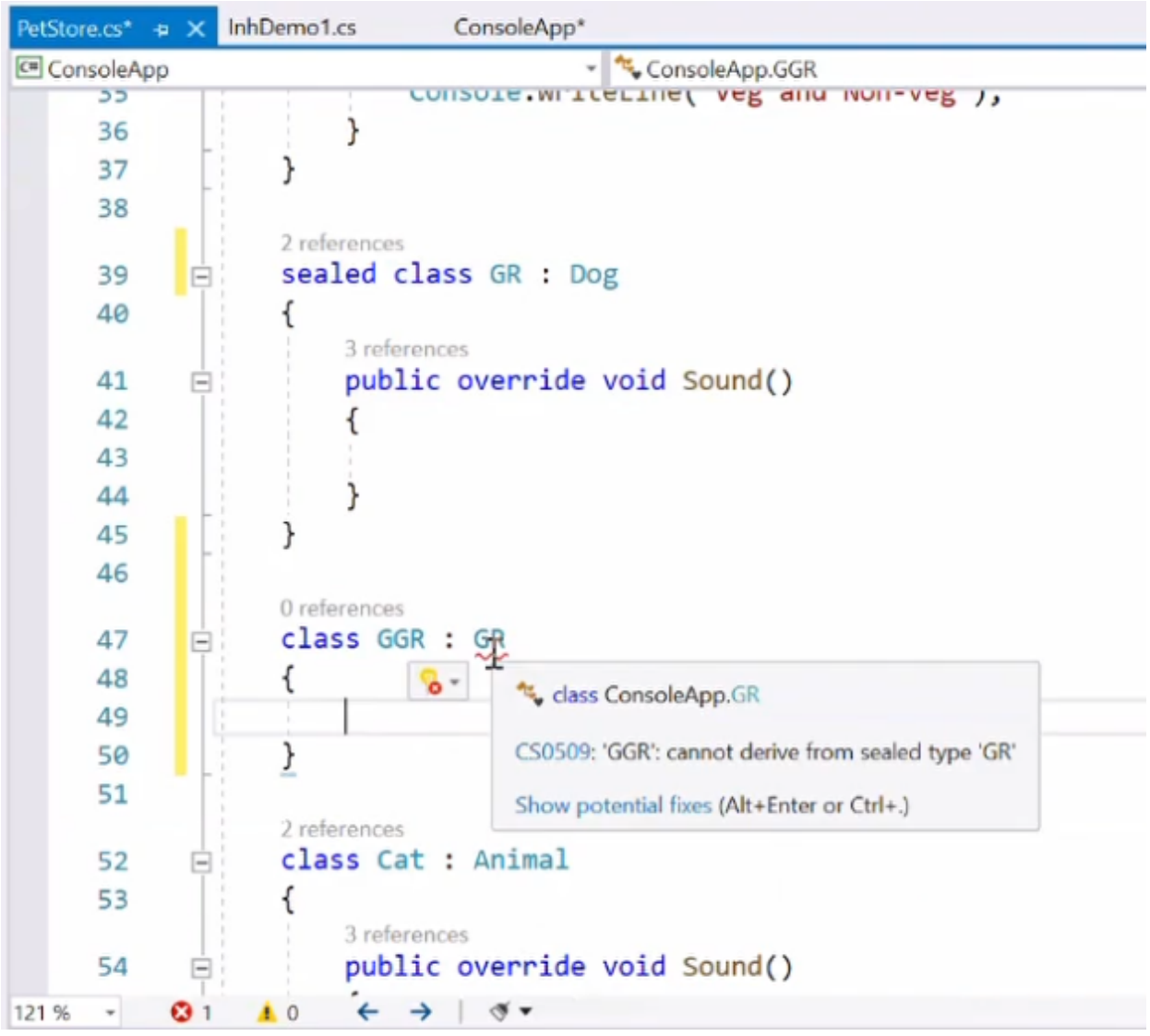
and GR is a Derived class right


PetStore.cs
InhDemo1.cs
ConsoleApp*
ConsoleApp
ConsoleApp.GR
28 //}
29 }
2 references
30 abstract class Dog : Animal
31 {
32
3 references
33 public override void Eat()
34 {
35 Console.WriteLine("Veg and Non-veg");
36 }
37 }
38
1 reference
39 class GR : Dog
40 {
3 references
41 public override void Sound()
42 {
43
44 }
45 }
46
2 references
47 class Cat : Animal
121 %
No issues found
Output
Package Manager Console
Error List
Immediate Window

GR is a breed right, So can you further divide GR into some other breed ?? -------- No
So I don't want somebody to Inherit because it is the smallest class what I have
So what I can do is
---------I can Seal it


PetStore.cs*
InhDemo1.cs
ConsoleApp*
ConsoleApp
ConsoleApp.GR
Sound
Console.WriteLine("Veg and Non-Veg");
}
}
1 reference
sealed class GR : Dog
{
3 references
public override void Sound()
{
}
}
2 references
class Cat : Animal
{
3 references
public override void Sound()
{
Console.WriteLine("Meo..Meo...");
}
3 references
public override void Eat()
121 %
No issues found
Output
Package Manager Console
Error List ...
Immediate Window
Ready
Type here to search

now If I try to inherit it, it will give me error



Which Class can be a Concrete Class?

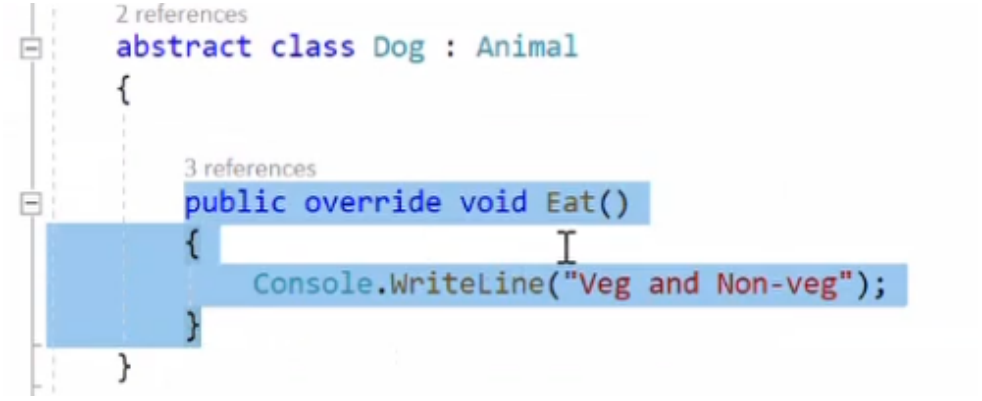
----Concrete classes are those, which will not have abstract methods in it

this will be the completely down classes, which will have all the information in it

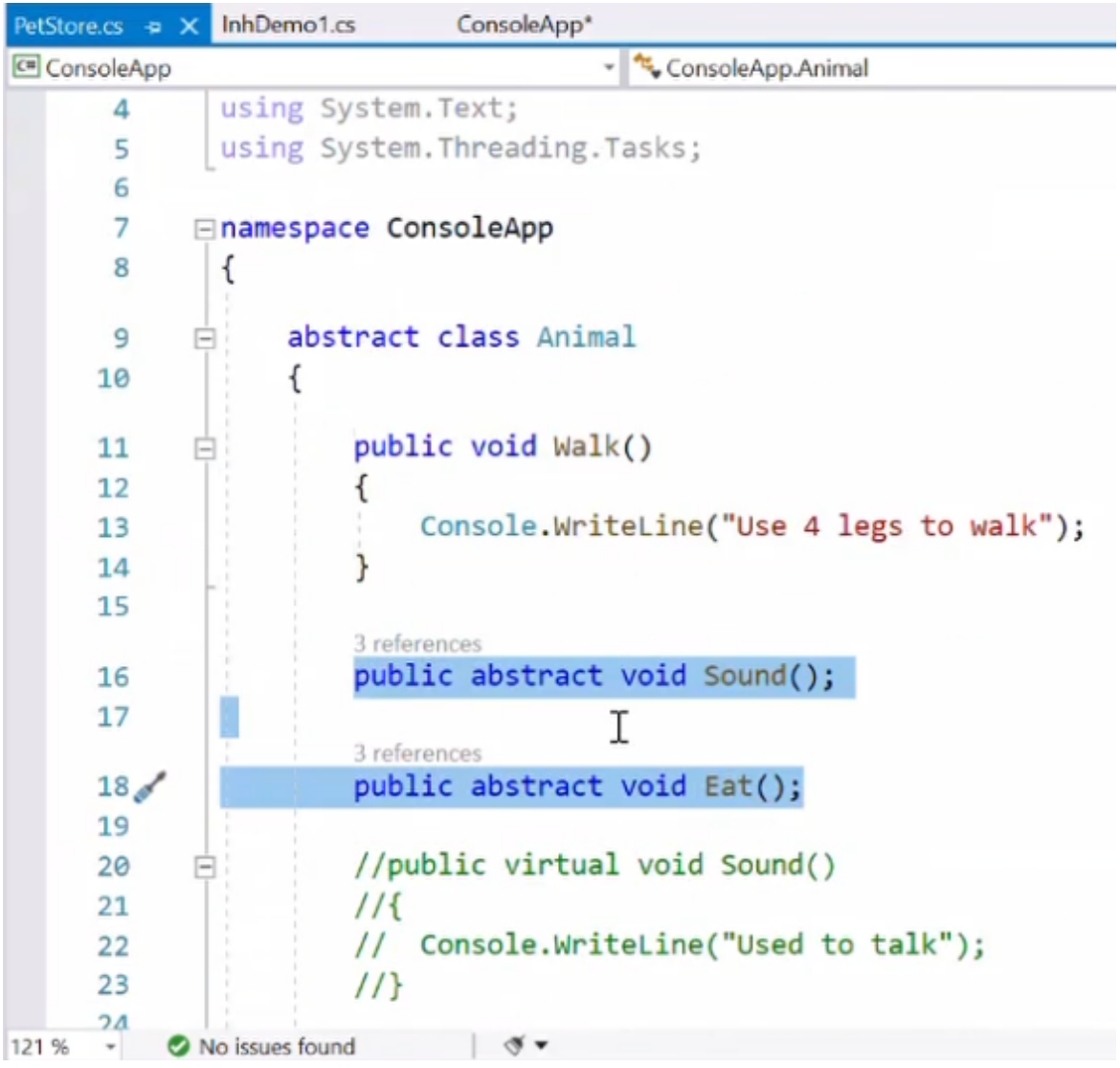
What are Abstract Classes ?

Abstract classes may or may not have a Implementation

in our case abstract class has the implementation

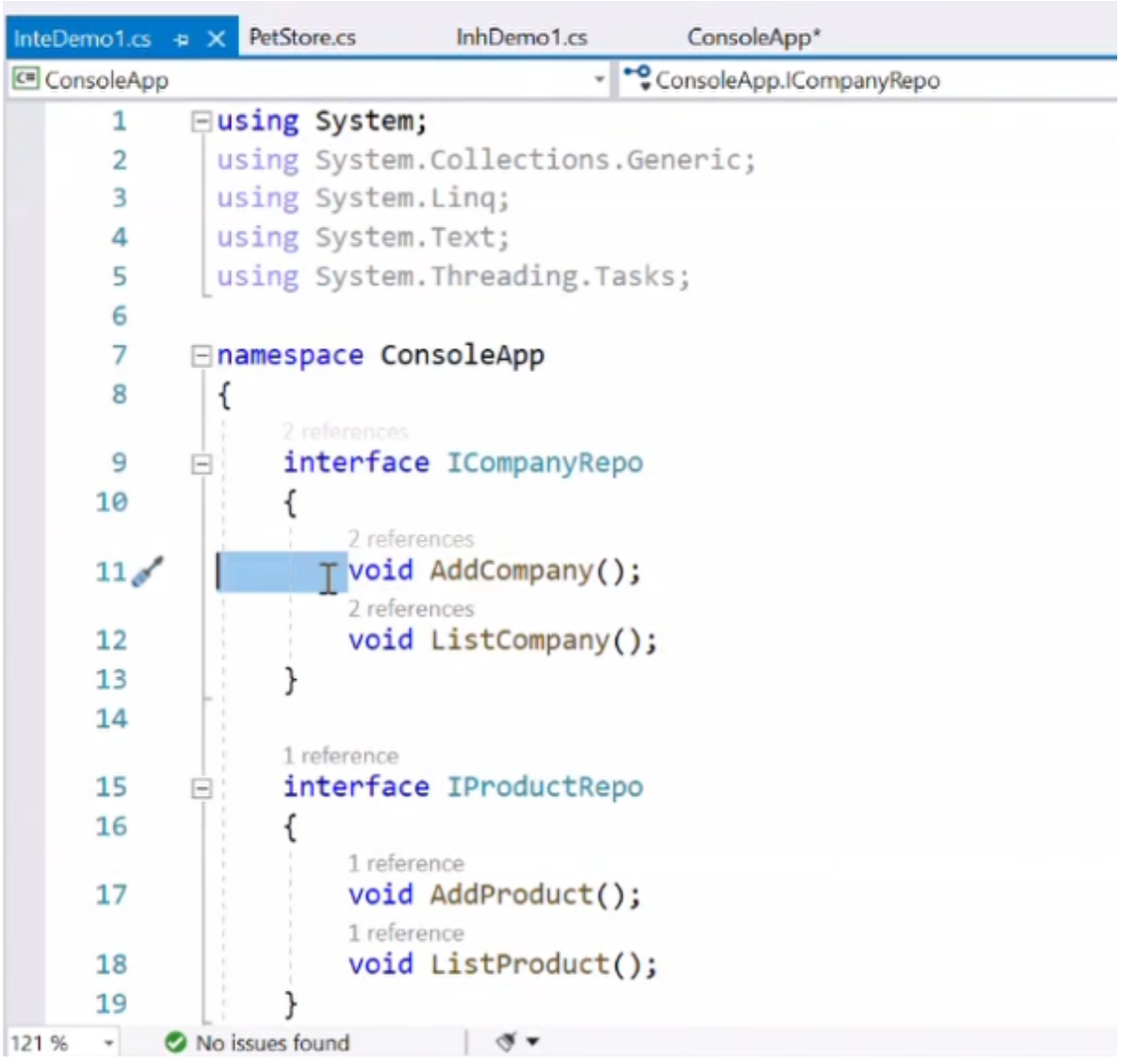


Look at this, it has a implementation and it has a abstract methods as well


PetStore.cs
InhDemo1.cs
ConsoleApp*
ConsoleApp
ConsoleApp.Animal
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp
{
abstract class Animal
{
public void Walk()
{
Console.WriteLine("Use 4 legs to walk");
}
3 references
public abstract void Sound();
3 references
public abstract void Eat();
//public virtual void Sound()
//{
// Console.WriteLine("Used to talk");
//}
121 %
No issues found

Now What are Interfaces ?
They are purely abstract , Remember the word abstract ------- Abstract means Dhundhla(Which is not clear to me, it can be half clear or completely blank)

Interfaces will have only abstract methods in it
Whenever you declare a interface, look at this I'm not specifying private, public, protected



So you should not, you cannot specify any access specifier here

and a**ll methods are Public by default in Interface**

we'll play a game, where you have to tell weather it is a **Interface** or **Abstract Class** or **Concreate Class**

Tree
(**Interface**)
Explanation: When I say tree what pitchure is comming in your mind ?
So we don't know each one person can draw a different image of tree
The Implementation we really don't know So this is abstract.

Chapati
(Concerte Class)
Explanation : You can made it of any shape, thin or thick
but you love it to be Round and it is made of same aata , so it comes in concerte class
because all of us will draw the same diagram of chapati

Rose
(Abstract Class)
Explanation: because there can be different types/colors of roses
Red, white, pink..... all of us can draw the rose but it can be of different colors
So Rose can be a concerate class by passing a parameter color as red

DOSA
(Abstract Class)
Explanation: When you go to hotel and ask for chapati, weather he ask you what kind of chaapati you want round or not, thin or not.....
No right
But when you ask for dosa, defenately he will ask, what kind of dosa you want (masala dosa, paneer dosa.....) So it is abstract, because the base which we use there will be common for all sorts of dosa, the batter will be same

Pen
(Abstract)
Explanation : because It can be ink pen, ball pen, gel pen .....

So From now when ever you see any thing, look for weather it is Interface or abstract or concreate
In our home most of thing we saw are abstract , you say its a laptop but i'll say Asus vivobook R542UQ
So it differes from model to mode

So this is called Attention to detailing, which is very important in our industry

This is 1 of the 4 thumb rule, which you have to follow