

C# - 11

Interface

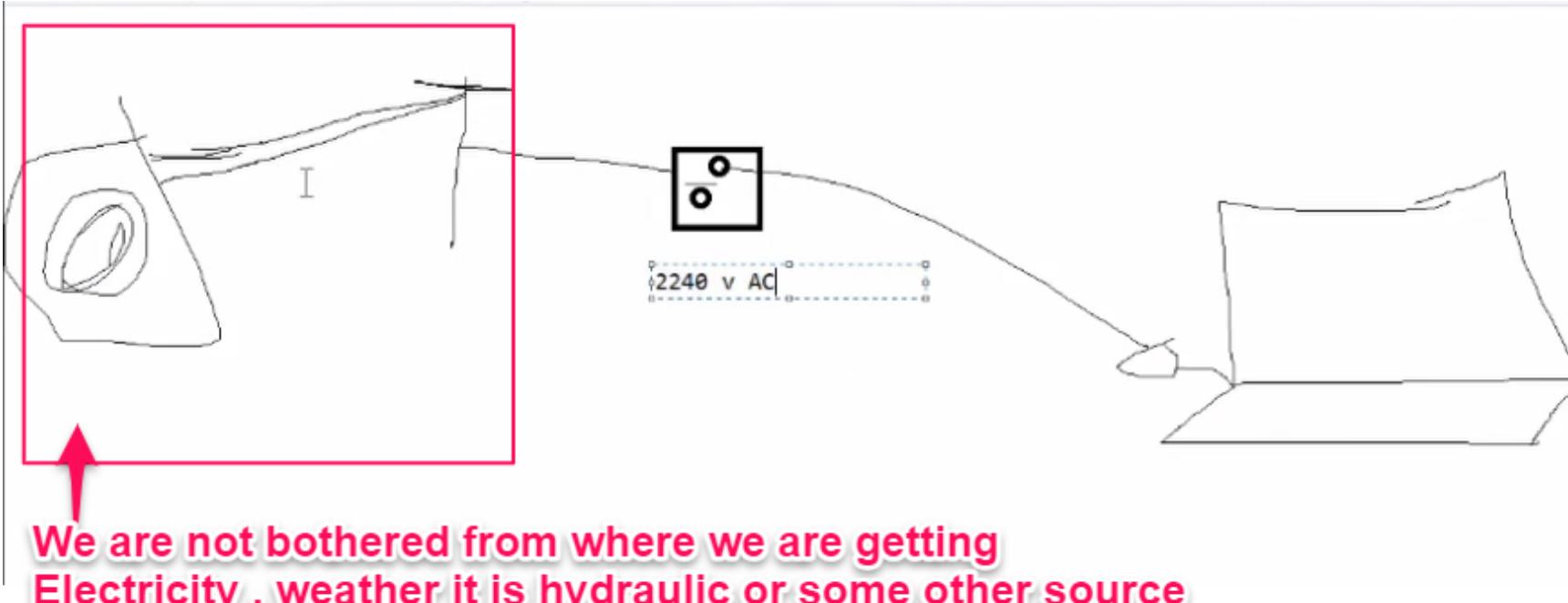
Interface

- Interface is a pure abstract class. That means, all the members of an interface are pure abstract.
- It does not contain any member with implementation
- interfaces are for realization relationship.
- A Realization is a relationship between two elements, in which one element (the client) realizes the behavior that the other element (the supplier) specifies.
- Objects of interface can't be created.

In General Interface is to Identify the behaviours

Suppose I have a TV

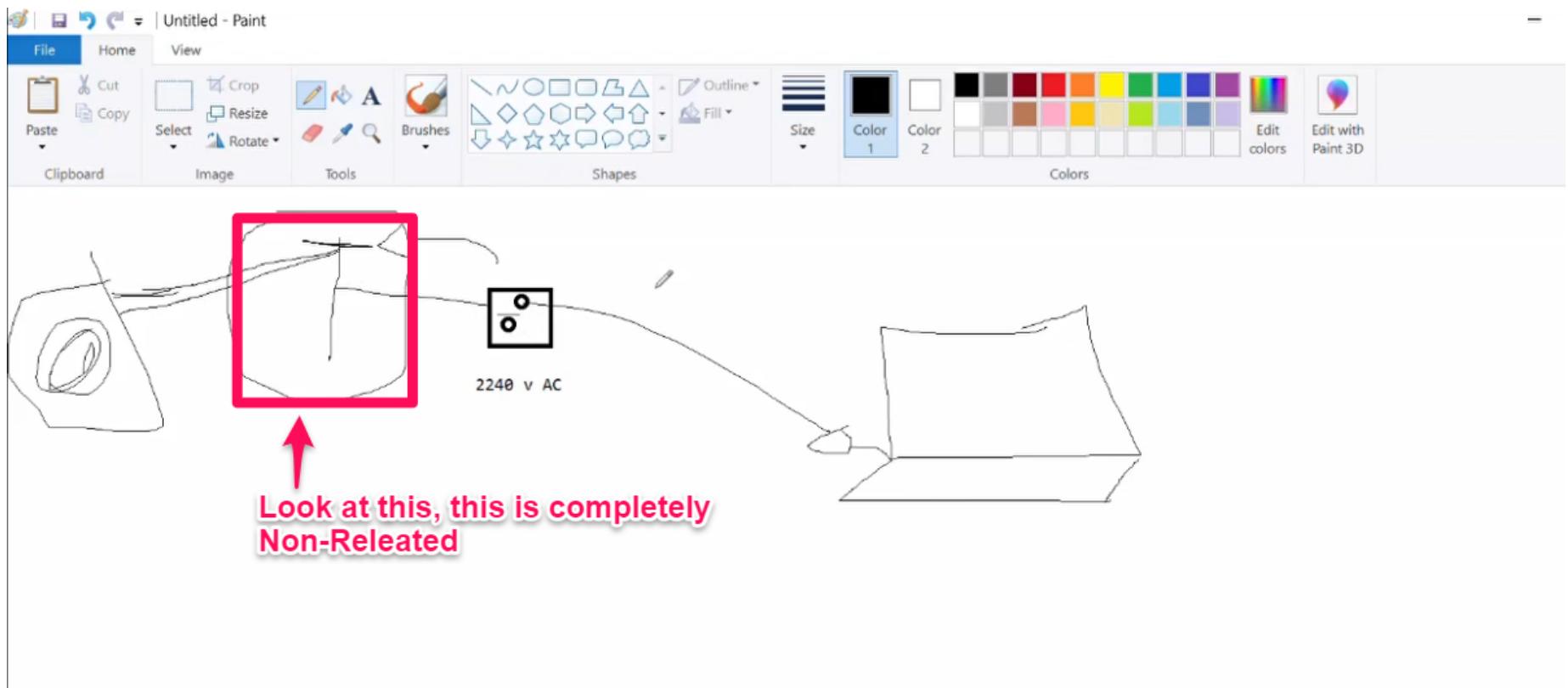
When you Count TV as a TV -----When it does the job of a TV , when it looks like a TV
we have a laptop with us, it has a adaptor which connects to 3 pin socket, if you are connecting to 2 pin pls change it to 3 pin
other wise your harddisk will corrupt, make sure it connects to ground



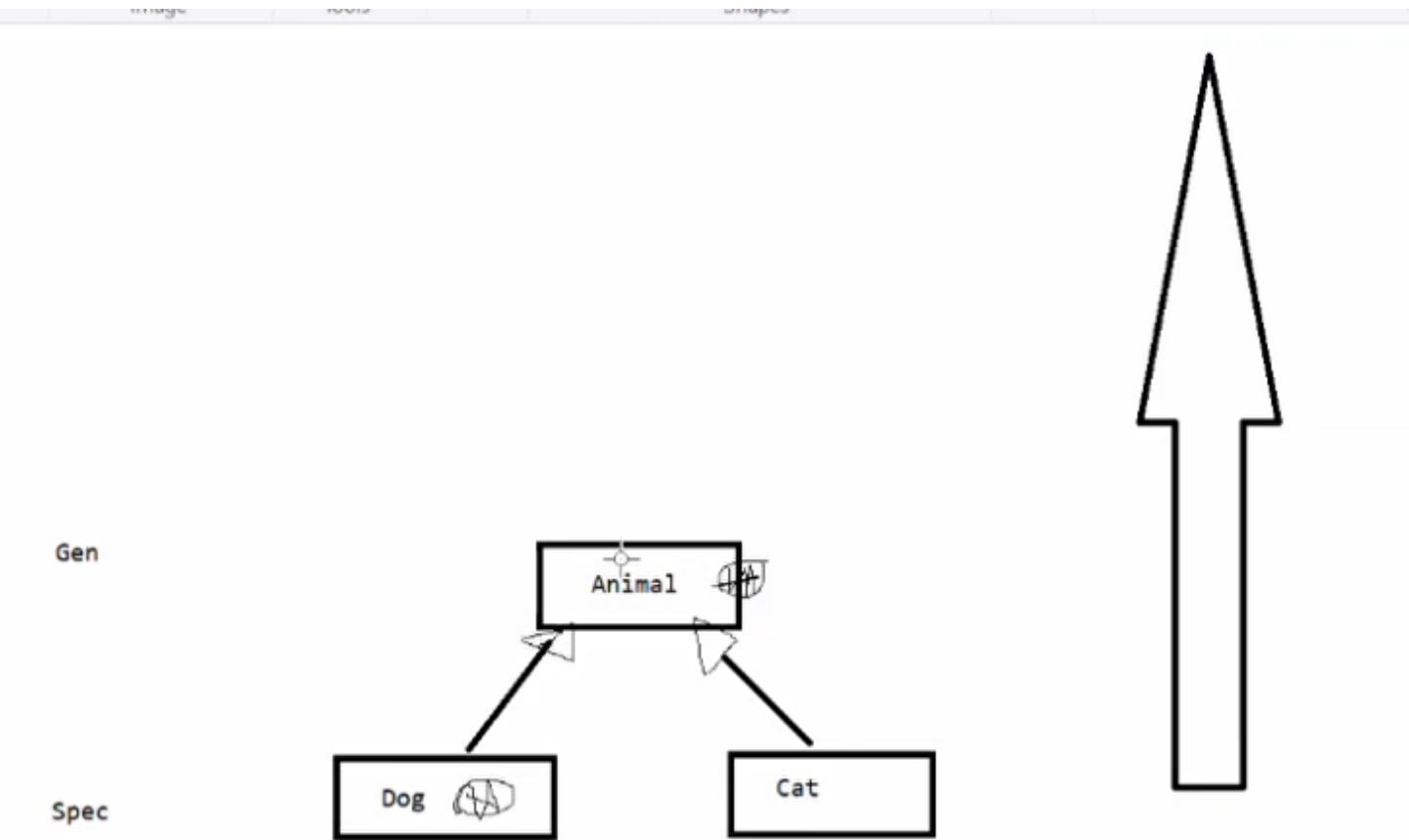
We are not bothered from where we are getting Electricity , weather it is hydraulic or some other source

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

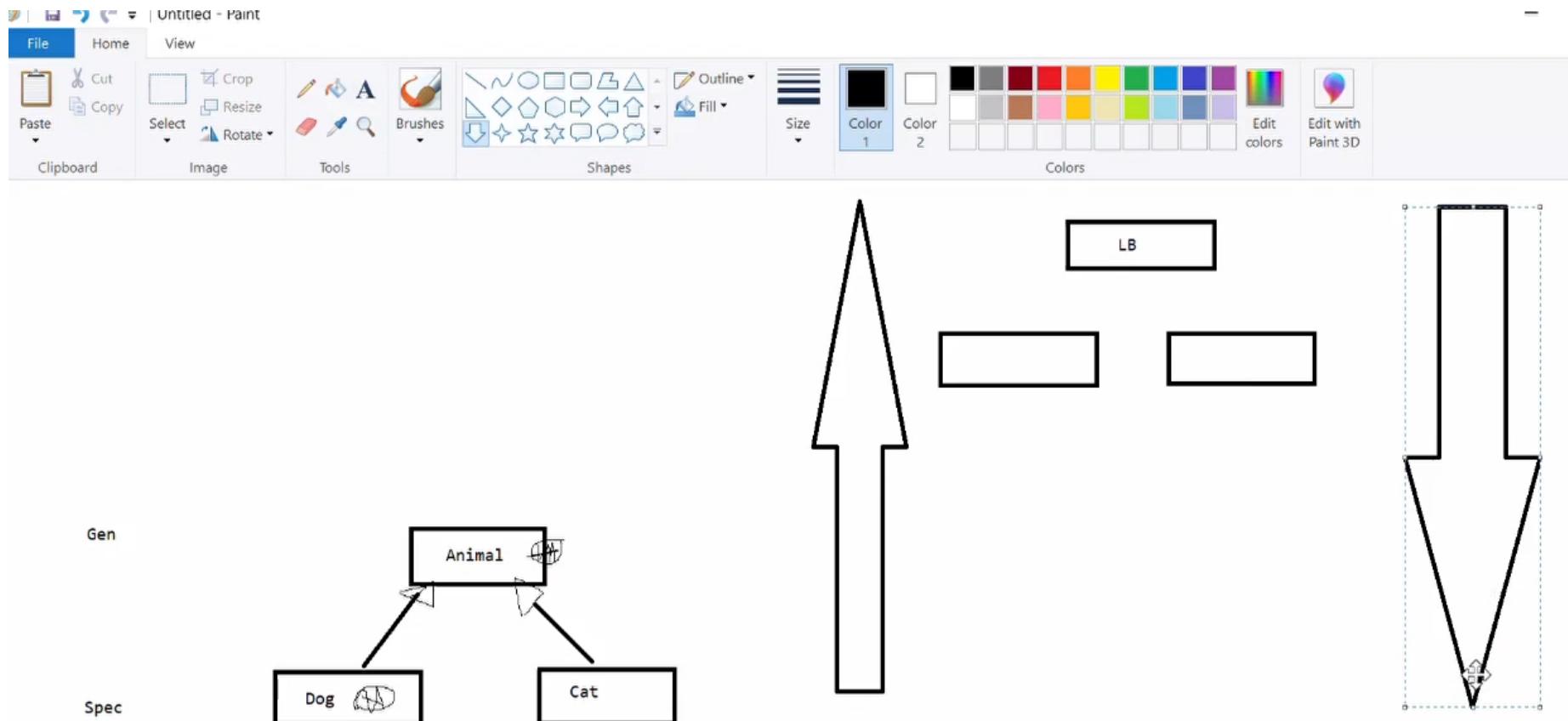
So 1st We'll identify all the behaviour down the line then we'll start implementing them

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

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays "PetStore.cs" and "ConsoleApp". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar. The toolbar contains various icons for file operations like Open, Save, and Build.

The code editor window shows the following C# code:

```
7  namespace ConsoleApp
8  {
9      abstract class Animal
10     {
11         public void Walk()
12         {
13             Console.WriteLine("Use 4 legs to walk");
14         }
15
16         3 references
17         public abstract void Sound();
18
19         3 references
20         public abstract void Eat();
21
22         //public virtual void Sound()
23         //{
24             // Console.WriteLine("Used to talk");
25         //}
26
27         //public virtual void Eat()
28         //{
29             // Console.WriteLine("Gives energy");
30         //}
31     }
32 }
```

The status bar at the bottom indicates "121 %", "No issues found", and the current tab is "Output".

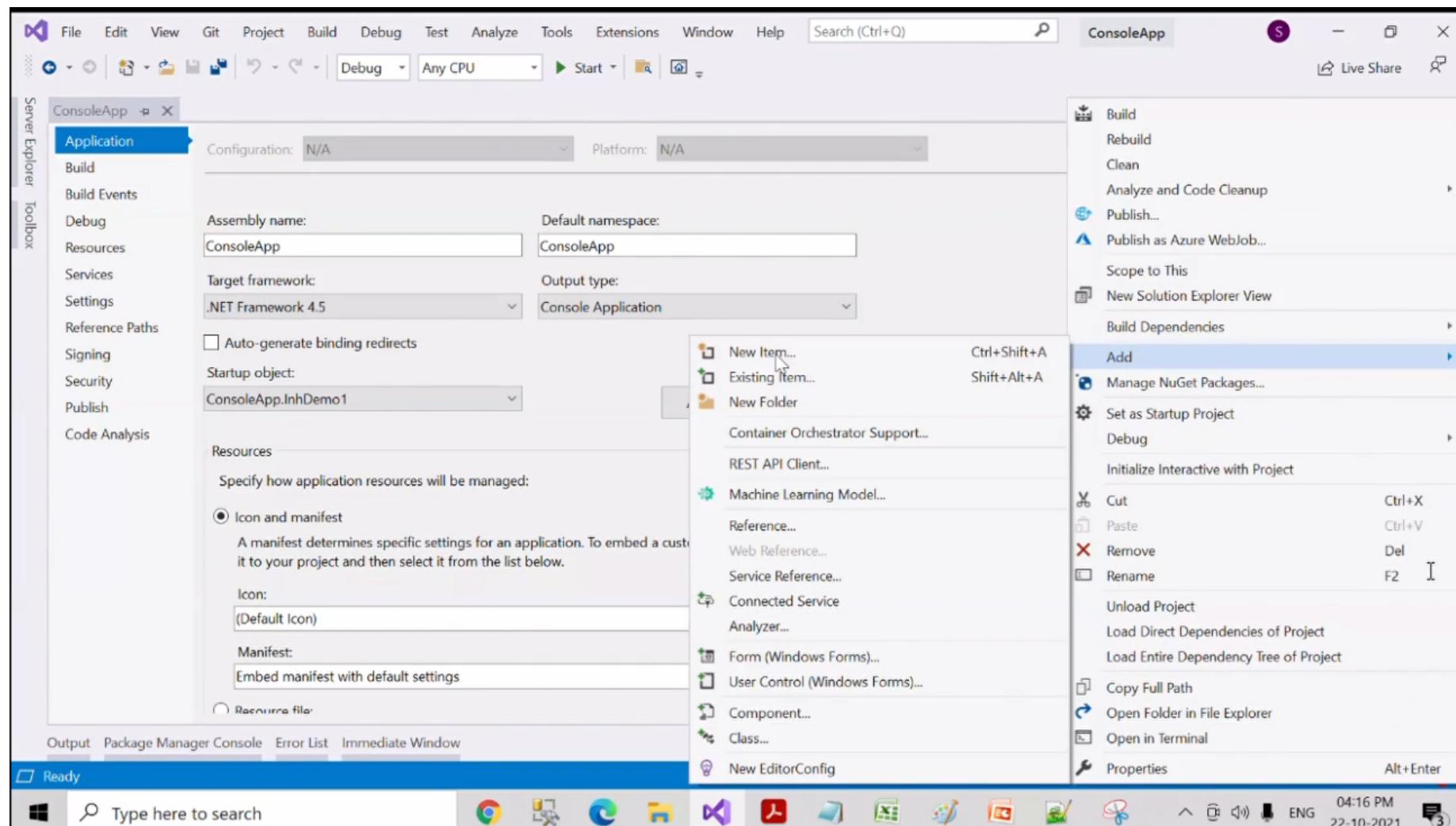
but tomorrow I want to introduce a new method and want to make sure only couple of classes should inherit it, So I cannot do that

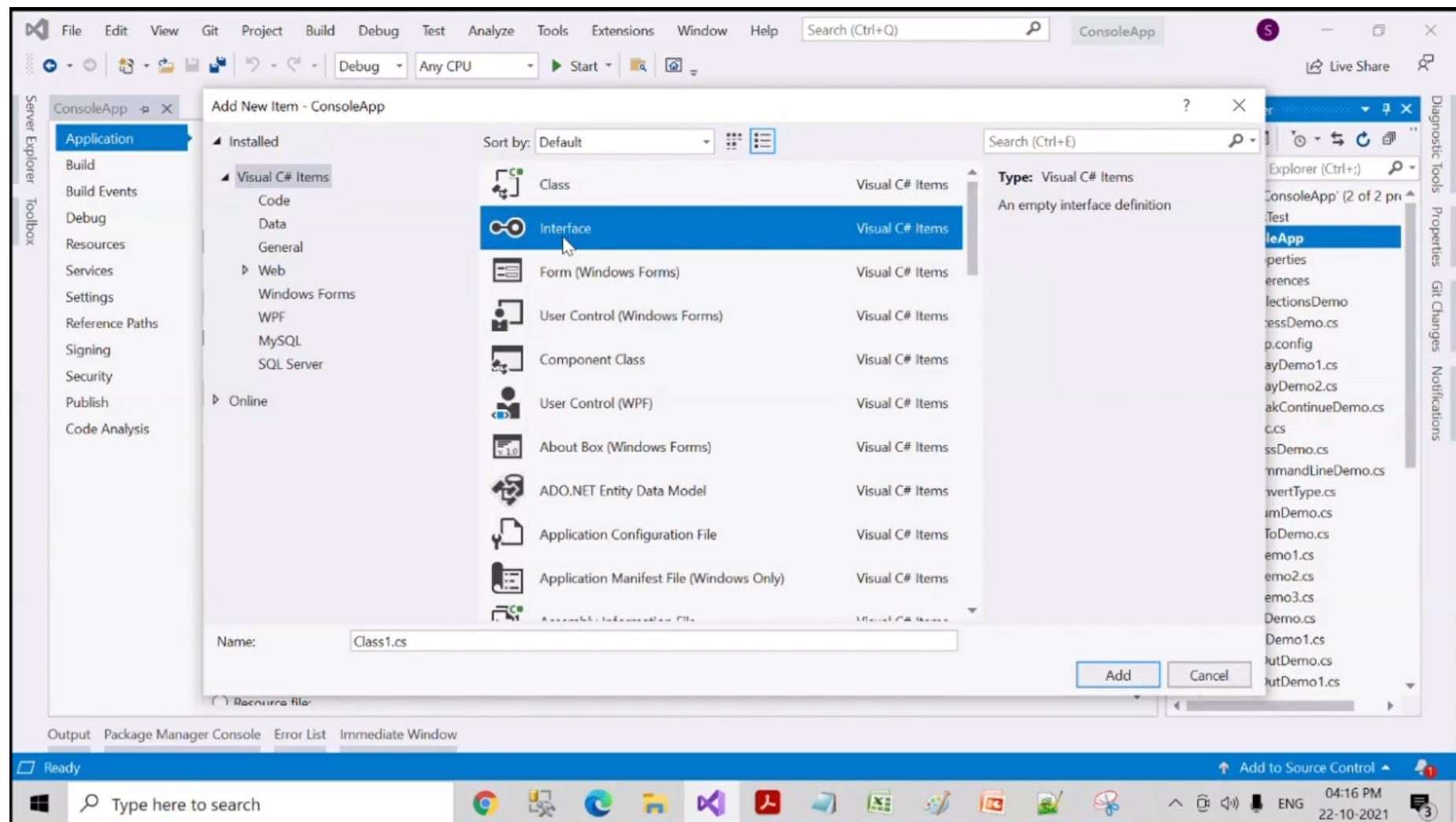
or

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

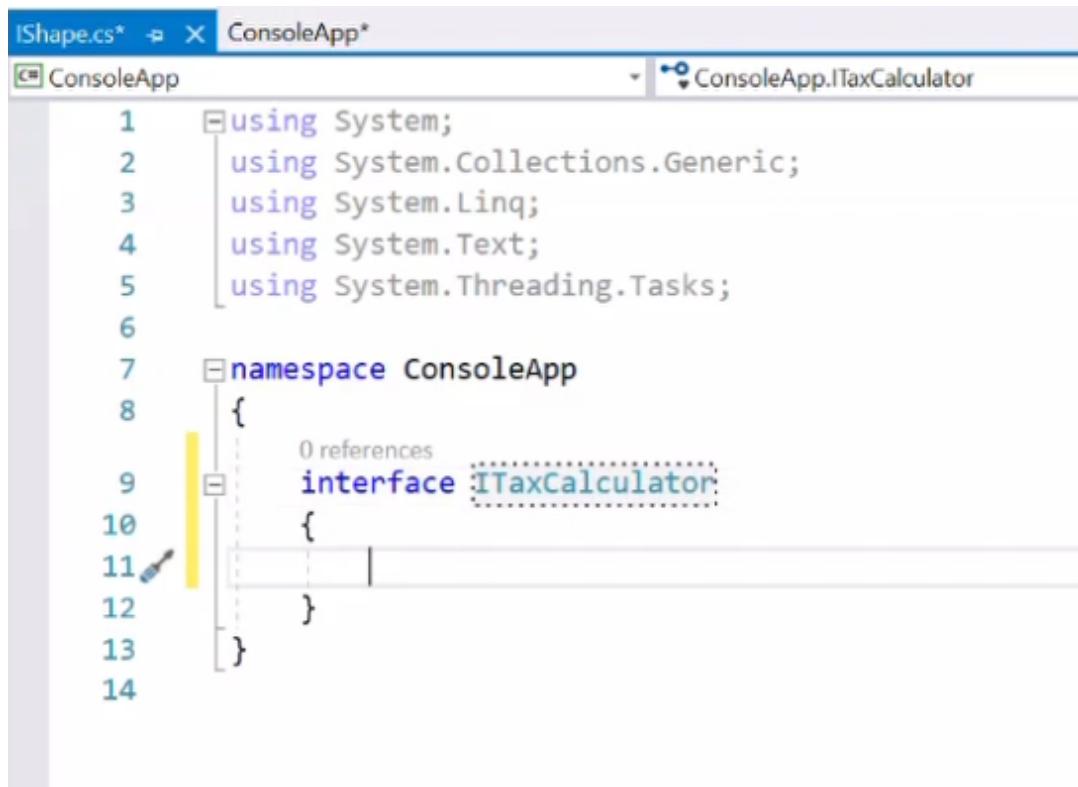
So we make sure that we not use **extends** here, In Java we say extends naa, we'll avoid using that as much as possible

Now whenever you want to create Interface follow this steps





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



The screenshot shows a code editor window with the file 'IShape.cs*' open. The code defines an interface named 'ITaxCalculator'. The code is as follows:

```
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      interface ITaxCalculator
10     {
11         // Method declarations
12     }
13 }
14
```

A yellow vertical bar highlights the word 'interface' in line 9, and a blue vertical bar highlights the identifier 'ITaxCalculator' in line 9. The status bar at the bottom of the editor shows the text 'this is the taxes'.

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

```
IShape.cs*  X  ConsoleApp*  
ConsoleApp  
ConsoleApp.ITaxCalculator  
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      interface ITaxCalculator  
10     {  
11         double CalculateTax(double baseSalary);  
12     }  
13 }  
14
```

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

The screenshot shows a Microsoft Visual Studio IDE window. The title bar indicates the project is named "ConsoleApp". The code editor displays the following C# code:

```
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      interface ITaxCalculator
10     {
11         double CalculateTax(double baseSalary);
12     }
13
14     class OldTaxCalculator : ITaxCalculator
15     {
16     }
17 }
18
19
```

A yellow vertical bar highlights the opening brace of the `OldTaxCalculator` class definition at line 14. A tooltip box is visible near the cursor, showing the text "1 reference" above the word `ITaxCalculator`. The cursor is positioned at the end of the `ITaxCalculator` interface name.

now as soon as I press CTRL + .

The screenshot shows a code editor in Visual Studio with the following code:

```
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      interface ITaxCalculator
10     {
11         double CalculateTax(double baseSalary);
12     }
13
14     class OldTaxCalculator : ITaxCalculator
15     {
16         // Implementation code here
17     }
18
19 }
```

A context menu is open at the end of the line '14' (before the colon), with the following options:

- Implement interface
- Implement all members explicitly
- Add accessibility modifiers
- Generate constructor 'OldTaxCalculator()'
- Suppress or Configure issues

The 'Implement interface' option is highlighted.

and click on Implement Interface

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search field. Below the menu is a toolbar with various icons. The left sidebar features the Server Explorer and Toolbox. The main workspace displays the code editor for a file named IShape.cs, which is part of a project named ConsoleApp. The code editor shows the following C# code:

```
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      interface ITaxCalculator
10     {
11         double CalculateTax(double baseSalary);
12     }
13
14     class OldTaxCalculator : ITaxCalculator
15     {
16
17         public double CalculateTax(double baseSalary)
18         {
19             throw new NotImplementedException();
20         }
21     }
}
```

A vertical yellow bar on the left margin indicates the current line of code being edited. The word "ITaxCalculator" is underlined with a red squiggle, suggesting it is a misspelling or a reference to an external type. The status bar at the bottom shows "121 %", "1", "0", and other standard status indicators.

i get this

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

IShape.cs* ✘ ConsoleApp*

ConsoleApp

ConsoleApp.OldTaxCalculator

CalculateTax(double baseSalary)

```
13
14     class OldTaxCalculator : ITaxCalculator
15     {
16         public double CalculateTax(double baseSalary)
17         {
18             double taxValue = 0;
19             if (baseSalary >= 250000 && baseSalary < 500000)
20             {
21                 taxValue = baseSalary * 0.05;
22             }
23             else if(baseSalary >= 500000 && baseSalary < 750000)
24             {
25                 taxValue = baseSalary * 0.2;
26             }
27             else if (baseSalary >= 750000 && baseSalary < 100000)
28             {
29                 taxValue = baseSalary * 0.2;
30             }
31             else if (baseSalary >= 100000 && baseSalary < 125000)
32             {
33                 taxValue = baseSalary * 0.2;
34             }
        }
```

121 %

Ln: 3 C

Output Package Manager Console Error List Immediate Window

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

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Project Name:** ConsoleApp.
- Toolbars:** Standard toolbar with icons for file operations, search, and help.
- Code Editor:** The main window displays the `IShape.cs` file under the `ConsoleApp` project. The code is as follows:

```
24     taxValue = baseSalary * 0.2;
25 }
26 else if (baseSalary >= 750000 && baseSalary < 1000000)
27 {
28     taxValue = baseSalary * 0.2;
29 }
30 else if (baseSalary >= 1000000 && baseSalary < 1250000)
31 {
32     taxValue = baseSalary * 0.3;
33 }
34 else if (baseSalary >= 1250000 && baseSalary < 1500000)
35 {
36     taxValue = baseSalary * 0.3;
37 }
38 else
39 {
40     taxValue = baseSalary * 0.3;
41 }
42
43 return taxValue;
44 }
45 }
46 }
```

- Solution Explorer:** Shows the solution structure with projects like `AccessTest`, `ConsoleApp`, and files such as `App.cs`, `Calc.cs`, etc.
- Status Bar:** Displays the current line (Ln: 43), character (Ch: 20), column (Col: 29), and tabs/crlf settings.
- Bottom Navigation:** Includes Output, Package Manager Console, Error List ..., Immediate Window, and a message bar indicating "Item(s) Saved".

now this is for old tax calculation

IShape.cs X ConsoleApp*

ConsoleApp

ConsoleApp.TaxCalc

Main()

```
11     double CalculateTax(double baseSalary);
12 }
13
14 class OldTaxCalculator : ITaxCalculator
15 {
16     public double CalculateTax(double baseSalary){...}
17 }
18
19 class TaxCalc
20 {
21     static void Main()
22     {
23         OldTaxCalculator taxCalculator = new OldTaxCalculator();
24         double taxValue = taxCalculator.CalculateTax(100000);
25         Console.WriteLine("Tax value is " + taxValue);
26     }
27 }
```

2 references

0 references

2 references

0 references

0 references

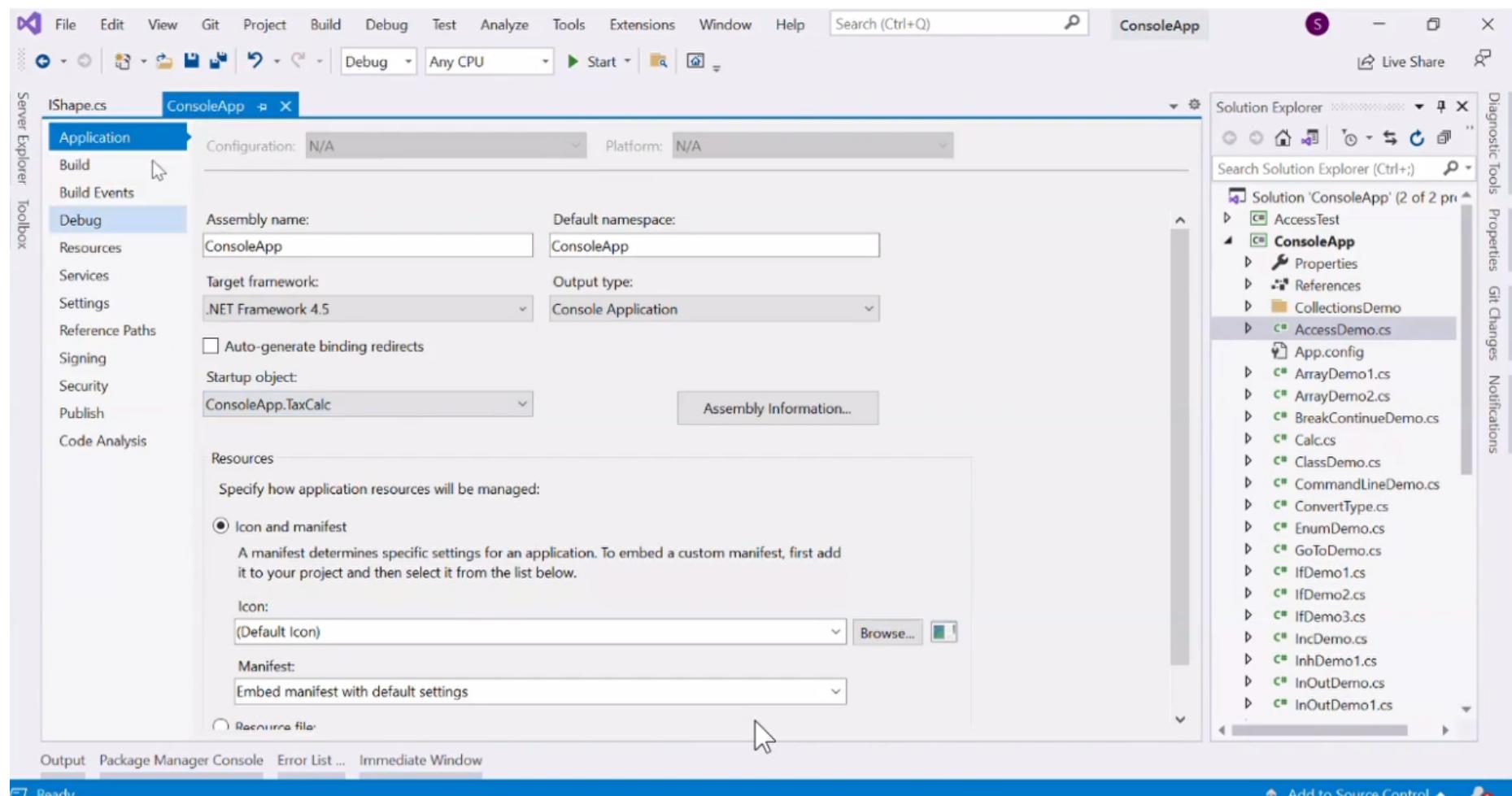
Ln: 53 Ch: 48 Col: 57 TABS CRLF

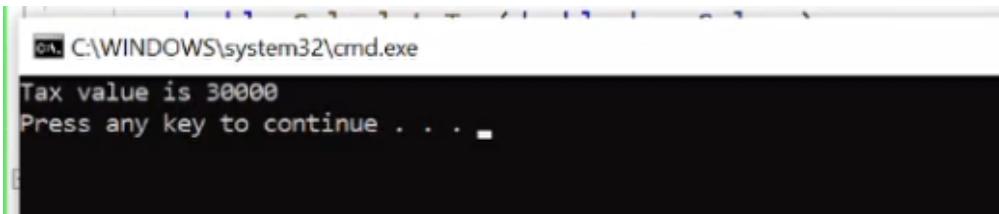
Output Package Manager Console Error List ... Immediate Window

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
- InhDemo1.cs
- InOutDemo.cs
- InOutDemo1.cs





A screenshot of a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.exe". The window contains the following text:
Tax value is 30000
Press any key to continue . . .

similarly we'll write for NewTaxCalculator

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

- Title Bar:** Shows "ConsoleApp" as the active project.
- Toolbars:** Standard Visual Studio toolbars for file operations, search, and navigation.
- Code Editor:** Displays the `ConsoleApp.cs` file content. The code implements an `ITaxCalculator` interface and provides three implementations: `OldTaxCalculator`, `NewTaxCalculator`, and a concrete implementation. The `NewTaxCalculator` class is highlighted with a green vertical bar on the left.
- Status Bar:** Shows "121 %", "No issues found", "Ln: 49 Ch: 30", and a cursor icon.
- Bottom Navigation:** Tabs for "Output", "Package Manager Console", "Error List", and "Immediate Window".

```
8 {
9     2 references
10    interface ITaxCalculator{...}
11
12    2 references
13    class OldTaxCalculator {...}
14
15    0 references
16    class NewTaxCalculator : ITaxCalculator
17    {
18        3 references
19        public double CalculateTax(double baseSalary)
20        {
21            double taxValue = 0;
22            if (baseSalary >= 250000 && baseSalary < 500000)
23            {
24                taxValue = baseSalary * 0.05;
25            }
26            else if (baseSalary >= 500000 && baseSalary < 750000)
27            {
28                taxValue = baseSalary * 0.2;
29            }
30            else if (baseSalary >= 750000 && baseSalary < 1000000)
31            {
32                taxValue = baseSalary * 0.25;
33            }
34        }
35    }
```

IShape.cs* X ConsoleApp

ConsoleApp

ConsoleApp.NewTaxCalculator

CalculateTax(double baseSalary)

```
double taxValue = 0;
if (baseSalary >= 250000 && baseSalary < 500000)
{
    taxValue = baseSalary * 0.05;
}
else if (baseSalary >= 500000 && baseSalary < 750000)
{
    taxValue = baseSalary * 0.1;
}
else if (baseSalary >= 750000 && baseSalary < 100000)
{
    taxValue = baseSalary * 0.015;
}
else if (baseSalary >= 1000000 && baseSalary < 1250000)
{
    taxValue = baseSalary * 0.2;
}
else if (baseSalary >= 1250000 && baseSalary < 1500000)
{
    taxValue = baseSalary * 0.25;
}
else
{
    taxValue = baseSalary * 0.3;
}
```

121 %

1 0

Ln: 70 Ch: 33 Col: 45

IShape.cs X ConsoleApp

ConsoleApp

ConsoleApp.TaxCalc

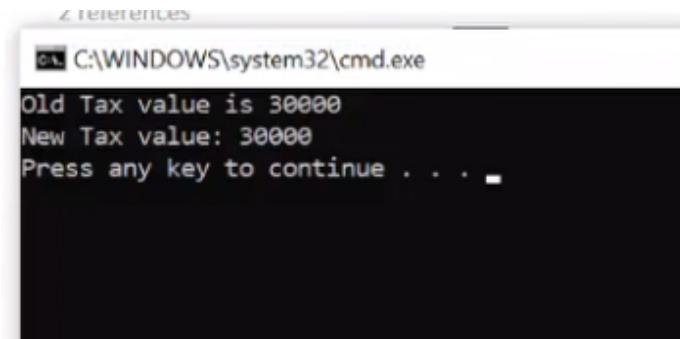
Main()

```
13
14     2 references
15     class OldTaxCalculator ...
16
17     2 references
18     class NewTaxCalculator ...
19
20     0 references
21     class TaxCalc
22     {
23         0 references
24         static void Main()
25         {
26             OldTaxCalculator taxCalculator = new OldTaxCalculator();
27             double taxValue = taxCalculator.CalculateTax(100000);
28             Console.WriteLine("Old Tax value is " + taxValue);
29
30             NewTaxCalculator taxCalculator1 = new NewTaxCalculator();
31             taxValue = taxCalculator1.CalculateTax(100000);
32             Console.WriteLine("New Tax value: " + taxValue);
33         }
34     }
35 }
```

121 %

No issues found

Ln: 86 Ch: 27 Col: 36



A screenshot of a Windows Command Prompt window titled "z references". The window shows the path "C:\WINDOWS\system32\cmd.exe". The text output is:
Old Tax value is 30000
New Tax value: 30000
Press any key to continue . . .

Now can you see that this is repeated code

IShape.cs X ConsoleApp

ConsoleApp

ConsoleApp.TaxCalc

Main()

```
13
14     2 references
15     class OldTaxCalculator ...
16
17     2 references
18     class NewTaxCalculator ...
19
20     0 references
21     class TaxCalc
22     {
23         0 references
24         static void Main()
25         {
26             OldTaxCalculator taxCalculator = new OldTaxCalculator();
27             double taxValue = taxCalculator.CalculateTax(1000000);
28             Console.WriteLine("Old Tax value is " + taxValue);
29
30             NewTaxCalculator taxCalculator1 = new NewTaxCalculator();
31             taxValue = taxCalculator1.CalculateTax(1000000);
32             Console.WriteLine("New Tax value: " + taxValue);
33         }
34     }
35 }
```

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

```
ConsoleApp.TaxCalc
Main()
  1 reference
  class NewTaxCalculator ...
  0 references
  class TaxCalc
  {
    2 references
    static void DisplayTax(ITaxCalculator taxCalculator)
    {
      double taxValue = taxCalculator.CalculateTax(1000000);
      Console.WriteLine("Old Tax value is " + taxValue);
    }
    0 references
    static void Main()
    {
      ITaxCalculator taxCalculator = new OldTaxCalculator();
      DisplayTax(taxCalculator);           I
      taxCalculator = new NewTaxCalculator();
      DisplayTax(taxCalculator);
    }
  }
}
```

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

The screenshot shows a Microsoft Visual Studio IDE interface. The main window displays a code editor with the file 'InteDemo1.cs' open. The code defines two interfaces: `ICompanyRepo` and `IProductRepo`, both containing methods `AddCompany()` and `ListCompany()`. The code editor uses color-coded syntax highlighting and Intellisense to highlight the interface names and method signatures. The Solution Explorer on the right shows a solution named 'ConsoleApp' with various files listed under the 'ConsoleApp' project.

```
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      interface ICompanyRepo
10     {
11         void AddCompany();
12         void ListCompany();
13     }
14
15     interface IProductRepo
16     {
17         void AddProduct();
18         void ListProduct();
19     }
20 }
```

then we write ShoponRepo class and inherit both the interface here

InteDemo1.cs ➔ X ConsoleApp*

ConsoleApp

0 references

```
21 class ShoponRepo : ICompanyRepo, IProductRepo
22 {
23     1 reference
24         public void AddCompany()
25         {
26             Console.WriteLine("Add company called");
27         }
28     1 reference
29         public void AddProduct()
30         {
31             Console.WriteLine("Add product called");
32         }
33     1 reference
34         public void ListCompany()
35         {
36             Console.WriteLine("List company called");
37         }
38     1 reference
39         public void ListProduct()
40         {
41             Console.WriteLine("List product called");
42         }
}
```

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 Control

Solution Explorer

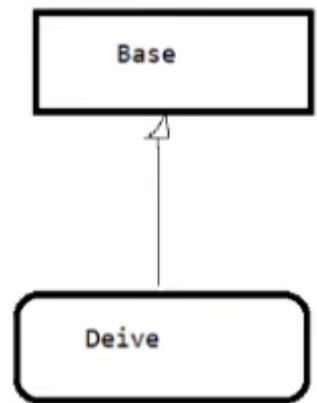
Search Solution Explorer (Ctrl+F)

Solution 'ConsoleApp' (2)

- AccessTest
- ConsoleApp
 - Properties
 - References
 - CollectionsDemo
 - AccessDemo.cs
 - App.config
 - ArrayDemo1.cs
 - ArrayDemo2.cs
 - BreakContinueDemo
 - Calc.cs
 - ClassDemo.cs
 - CommandLineDemo
 - 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

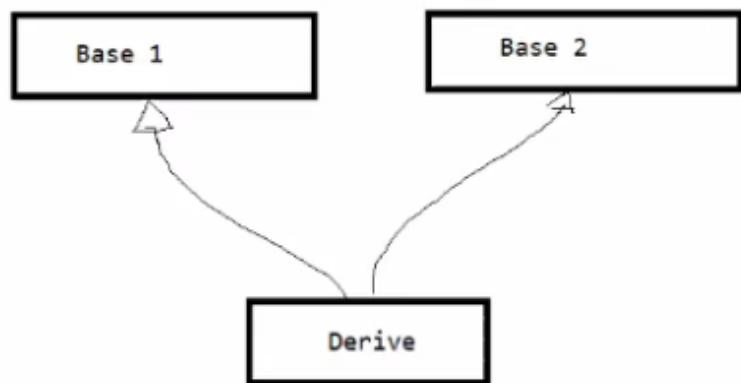


2. Multiple Inheritance - 2 base class and 1 Derived class (**Java and C# dosen't support Multiple Inheritance**) if you have to do something like this, you have to achieve this through Interface

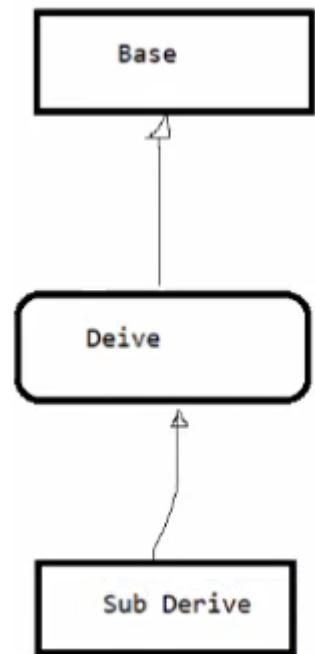
```
18     void ListProduct();
19 }
20
21 class ShoponRepo : ICompanyRepo, IProductRepo
22 {
23     public void AddCompany()
24     {
25         Console.WriteLine("Add company called");
26     }
27
28     public void AddProduct()
```

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



The screenshot shows the Visual Studio IDE with the code editor open to `InteDemo1.cs`. The code implements the `ICompanyRepo` interface and uses `ShoponRepo` as a concrete implementation. The code is annotated with callouts and highlights:

- Line 9:** `interface ICompanyRepo` has a callout indicating "2 references".
- Line 15:** `interface IPromoteRepo` has a callout indicating "1 reference".
- Line 21:** `class ShoponRepo` has a callout indicating "1 reference".
- Line 44:** `class InteDemo1` has a callout indicating "0 references".
- Line 46:** `static void Main()` has a callout indicating "0 references".
- Line 48:** `ICompanyRepo companyRepo = new ShoponRepo();` has a yellow highlight under "companyRepo" and "ShoponRepo".

```
8     {
9         [+] 2 references
10        interface ICompanyRepo ...
11
12        [+] 1 reference
13        interface IPromoteRepo ...
14
15        [+] 1 reference
16        class ShoponRepo ...
17
18
19
20
21        [+] 0 references
22        class InteDemo1
23
24        {
25            [+] 0 references
26            static void Main()
27            {
28                ICompanyRepo companyRepo = new ShoponRepo();
29            }
30
31        }
32
33    }
```

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
3     public void ListProduct()
4     {
5         Console.WriteLine("List product called");
6     }
7
8 0 references
9     class InteDemo1
10    {
11        0 references
12        static void Main()
13        {
14            ICompanyRepo companyRepo = new ShoponRepo();
15            companyRepo.|
```

The screenshot shows a code editor in Visual Studio displaying C# code. A tooltip is open over the `AddCompany` method of the `ICompanyRepo` interface, listing its members: `AddCompany`, `Equals`, `GetHashCode`, `GetType`, `ListCompany`, and `ToString`. The code being edited is part of the `Main` method of the `InteDemo1` class, which creates an instance of `ShoponRepo` and calls its `AddCompany` method.

```
InteDemo1.cs*  ConsoleApp*  
ConsoleApp  
ConsoleApp.InteDemo1  
Main()  
2 references  
33     public void ListCompany()  
34     {  
35         Console.WriteLine("List company called");  
36     }  
37  
1 reference  
38     public void ListProduct()  
39     {  
40         Console.WriteLine("List product called");  
41     }  
42 }  
43  
0 references  
44     class InteDemo1  
45     {  
46         static void Main()  
47         {  
48             ICompanyRepo companyRepo = new ShoponRepo();  
49             companyRepo.AddCompany();  
50             companyRepo.ListCompany();  
51         }  
52     }  
121 %  No issues found  
Output  Package Manager Console  Error List  Immediate Window  
Item(s) Saved  
Type here to search
```

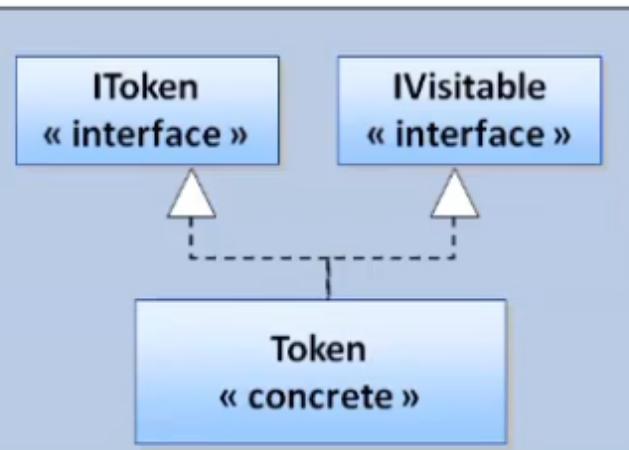
Where is AddProduct and ListProduct ??

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

```
class Token: IToken, IVisitable
{
    public string Name()
    {
        ...
    }

    public void Accept(IVisitor v)
    {
        ...
    }
}
```

Same access
Same return type
Same name
Same parameters

```
public virtual string Name()
{
    ...
}
```

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

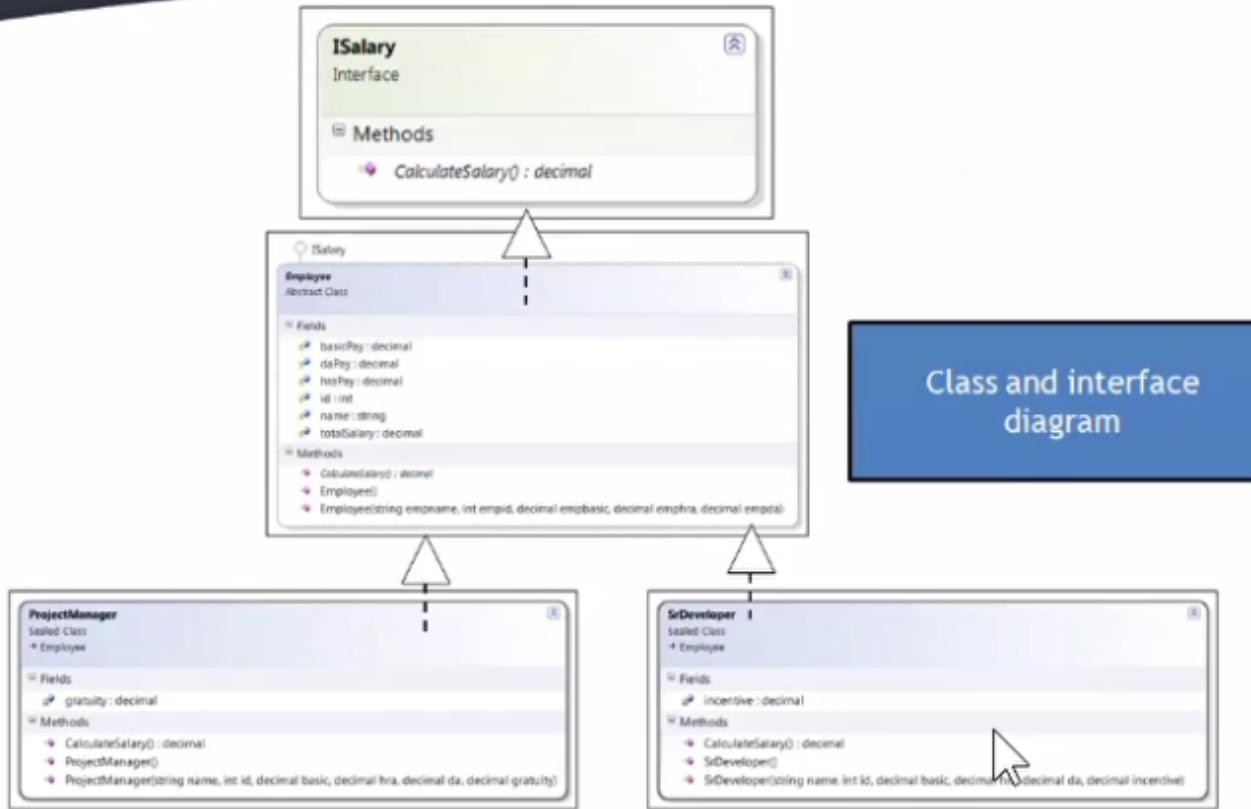
```
Token tokenobject = new Token();
Tokenobject.Name();
```

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



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.

```
public struct Employee
{
    public string firstName;
    public int age;
    public Employee()
    {
        firstName="Joseph";
        age=30;
    }
    public Employee(string fname,
                   int empAge)
    {
        firstName=fname;
        age=empAge;
    }
}
```

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

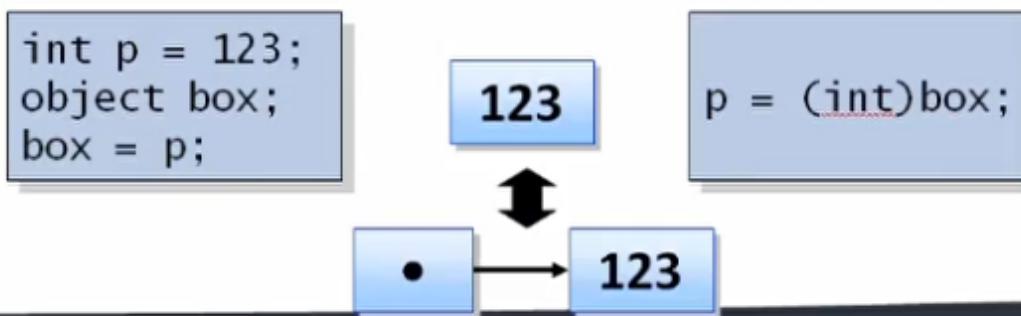
- The object type is the base for all classes
- Any reference can be assigned to object
- Any object variable can be assigned to any reference
 - With appropriate type conversion and checks
- The object type and is operator

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

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

Boxing and Unboxing

- Boxing: Converting value type into reference type
- Unboxing: Converting the reference type back to the same value type
 - While unboxing you have to convert the reference type back to the same data type first and then it can cast to any other data type (implicit).



What is the super class of this boxing class

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays a C# code editor with the file `BoxingDemo.cs` open. The code defines a `BoxingDemo` class within the `ConsoleApp` namespace. The `Main()` method contains a line of code that creates an instance of the `BoxingDemo` class. A yellow lightbulb icon is visible next to this line, indicating a potential issue. The Solution Explorer on the right shows the project structure with multiple files listed under the `ConsoleApp` folder.

```
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 BoxingDemo
10    {
11        static void Main()
12        {
13            BoxingDemo demo = new BoxingDemo();
14        }
15    }
16}
17
```

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

BoxingDemo.cs* ✘ X ConsoleApp*

ConsoleApp

ConsoleApp.BoxingDemo

Main()

```
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 BoxingDemo : Object
10     {
11         static void Main()
12         {
13             BoxingDemo demo = new BoxingDemo();
14         }
15     }
16 }
17
```

121 % No issues found

Output Package Manager Console Error List ... Immediate Window

Item(s) Saved

Type here to search

Chrome Task View Taskbar

Back Forward Home Stop

So Object is the super classes of all this classes

What is there in my Object class

BoxingDemo.cs* ✘ X ConsoleApp*

ConsoleApp

ConsoleApp.BoxingDemo

Main()

```
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      2 references
10     class BoxingDemo : Object
11     {
12         0 references
13         static void Main()
14         {
15             BoxingDemo dem
16         }
17     }
}
```

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+.)

The screenshot shows a code editor in Visual Studio displaying the `Object` class from the `mscorlib` assembly. The code is color-coded, with `public`, `private`, and `protected` keywords in blue, and `Object` itself in red. The code includes several static methods like `Equals` and `ReferenceEquals`, and virtual methods like `GetHashCode` and `GetType`. A tooltip at the top right indicates the code is "Object [from metadata]". The status bar at the bottom shows "No issues found".

```
Assembly mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
using ...
namespace System
{
    public class Object
    {
        public Object();
        ~Object();

        public static bool Equals(Object objA, Object objB);
        public static bool ReferenceEquals(Object objA, Object objB);
        public virtual bool Equals(Object obj);
        public virtual int GetHashCode();
        public Type GetType();
        public virtual string ToString();
        protected Object MemberwiseClone();
    }
}
```

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 Destroy

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

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

- Title Bar:** Displays "BoxingDemo.cs*" and "ConsoleApp*".
- Solution Explorer:** Shows a project named "ConsoleApp" with a file "ConsoleApp.BoxingDemo".
- Code Editor:** Contains the following C# code:

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             string name = "Shashi";
14             BoxingDemo demo = new BoxingDemo();
15         }
16     }
17 }
18 }
```
- Toolbars and Status Bar:** Standard Visual Studio toolbars and status bar are visible at the bottom.

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

```
BoxingDemo.cs*  X  ConsoleApp*
```

```
ConsoleApp
```

```
ConsoleApp.BoxingDemo
```

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             string name = "Shashi";
14             name = 1234; // Warning icon here
15             BoxingDemo demo = new BoxingDemo();
16         }
17     }
18 }
```

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

BoxingDemo.cs* ✘ X ConsoleApp*

ConsoleApp

ConsoleApp.BoxingDemo

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             string name = "Shashi";
14             //name = 1234;
15
16             Object obj = "Shashi";
17             obj = 1234;
18
19             BoxingDemo demo = new BoxingDemo();
20         }
21     }
22 }
```

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* ✘ X ConsoleApp*

ConsoleApp

ConsoleApp.BoxingDemo

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             string name = "Shashi";
14             //name = 1234;
15
16             Object obj = "Shashi";
17             string s = obj;
18
19             BoxingDemo demo = new BoxingDemo();
20         }
21     }
22 }
```

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**

The screenshot shows a Microsoft Visual Studio code editor window for a C# file named `BoxingDemo.cs`. The code defines a `BoxingDemo` class with a `Main` method. In the `Main` method, a string variable `name` is assigned the value "Shashi". A comment indicates an alternative assignment: `//name = 1234;`. Below this, an object `obj` is created and assigned the value "Shashi" with a note: `//boxing`. The code editor uses color coding for keywords and variables, and includes a vertical navigation bar on the left side. The status bar at the bottom shows the zoom level (121%), error count (0), warning count (1), and navigation icons.

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             string name = "Shashi";
14             //name = 1234;
15
16             Object obj = "Shashi"; //boxing
17
18
19             BoxingDemo demo = new BoxingDemo();
20         }
21     }
22 }
```

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

BoxingDemo.cs* ✘ X ConsoleApp*

ConsoleApp

ConsoleApp.BoxingDemo

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             Object obj = "Shashi"; //boxing
14
15             string s = obj;
16
17             BoxingDemo demo = new BoxingDemo();
18         }
19     }
20 }
```

So you need to use predefined method .ToString()

BoxingDemo.cs* ✘ × ConsoleApp*

ConsoleApp

ConsoleApp.BoxingDemo

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             Object obj = "Shashi"; //boxing
14
15             string s = obj.ToString();
16
17             BoxingDemo demo = new BoxingDemo();
18         }
19     }
20 }
21
```

**So this Is CalledUnBoxing - 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

```
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 BoxingDemo
10     {
11         static void Main()
12         {
13             Object obj = "Shashi"; //boxing
14
15             string s = obj.ToString();
16
17             BoxingDemo demo = new BoxingDemo();
18         }
19     }
20 }
21
```

Scope

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

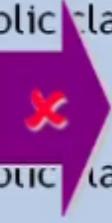
```
public class Bank
{
    public class Account
    {
        public void Deposit(decimal amount)
        {
            balance += amount;
        }
        private decimal balance;
    }
    public Account OpenAccount( ) { ... }
}
```

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

```
// From Vendor A
public class Widget { ... }           public class VendorAWidget
                                         { ... }

// From Vendor B
public class Widget { ... }           public class VendorBWidget
                                         { ... }
```



Declaring Namespaces

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

```
namespace VendorB  
{  
    public class Widget  
    { ... }  
}
```

```
namespace Microsoft  
{  
    namespace Office  
    {  
        ...  
    }  
}
```

```
namespace Microsoft.Office  
{  
}
```

shorthand

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(); X
        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 I'll imagine I'll create a static class and tried to put public method inside it

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `InhDemo1.cs`. The code defines a `ConsoleApp` namespace containing a `StaticClass` with a `Show()` method. It also includes declarations for `Base`, `Derive`, `DegreeStudent`, and `InhDemo1` classes. The Solution Explorer on the right lists various demo files, with `InhDemo1.cs` currently selected.

```
InhDemo1.cs*  X  ConsoleApp*  
ConsoleApp  
7  namespace ConsoleApp  
8  {  
9  }  
0 references  
10 static class StaticClass  
11 {  
12     0 references  
13     public void Show()  
14     {  
15     }  
16 }  
17  
3 references  
18 class Base...  
30  
3 references  
31 class Derive ...  
44  
1 reference  
45 class DegreeStudent ...  
55  
0 references  
56 class InhDemo1  
57 {  
121 %  X  1  0  ←  →  ⏪  ⏩  Ln: 14  Ch: 13  TABS  CRLF
```

Solution Explorer

- CollectionsDemo
- AccessDemo.cs
- App.config
- ArrayDemo1.cs
- ArrayDemo2.cs
- BoxingDemo.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
- IntDemo1.cs
- IntToWord.cs
- IntTypeDemo.cs
- ITaxCalcDemo.cs

look at this it gives me error

```
namespace ConsoleApp
{
    static class StaticClass
    {
        public void Show()
        {
            void StaticClass.Show()
        }
    }
}
```

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* X ConsoleApp*

ConsoleApp

ConsoleApp.StaticClass

Show()

```
7  namespace ConsoleApp
8  {
9
10 static class StaticClass
11 {
12     public static void Show()
13     {
14     }
15 }
16 }
17
18 class Base ...
19
20
21 class Derive ...
22
23
24 class DegreeStudent ...
25
26
27 class InhDemo1
28 {
```

121 % No issues found

Now can you inherit static class

----- No

InhDemo1.cs* X ConsoleApp*

ConsoleApp

ConsoleApp.Base

```
7  namespace ConsoleApp
8  {
9
10 static class StaticClass
11 {
12     public static void Show()
13     {
14     }
15 }
16 }
17
18 class Base : StaticClass
19 {
20     public void Show()
21     {
22         Show();
23     }
24
25     public void Show()
26     {
27     }
}
```

1 reference

1 reference

0 references

1 reference

3 references

CS0709: 'Base': cannot derive from static class 'StaticClass'

Show potential fixes (Alt+Enter or Ctrl+.)

No In Petstore we have Animal class as Abstract

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, and Window. Below the menu is a toolbar with various icons. The main window displays three tabs: PetStore.cs, InhDemo1.cs, and ConsoleApp*. The ConsoleApp* tab is active, showing the following C# code:

```
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      abstract class Animal
10     {
11         public void Walk()
12         {
13             Console.WriteLine("Use 4 legs to walk");
14         }
15
16         public abstract void Sound();
17
18         public abstract void Eat();
19
20         //public virtual void Sound()
21         //{
22         //    Console.WriteLine("Animal sound");
23         //}
```

The status bar at the bottom indicates "121 %", "No issues found", and shows tabs for Output, Package Manager Console, Error List ..., and Immediate Window.

and GR is a Derived class right

PetStore.cs X InhDemo1.cs ConsoleApp*

ConsoleApp

ConsoleApp.GR

```
28         //}
29     }
30     abstract class Dog : Animal
31     {
32
33         public override void Eat()
34         {
35             Console.WriteLine("Veg and Non-veg");
36         }
37     }
38
39     class GR : Dog
40     {
41         public override void Sound()
42         {
43
44         }
45     }
46
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

```
CONSOLE.WriteLine("Veg and Non-Veg"),  
36 }  
37 }  
38  
39 1 reference  
sealed class GR : Dog  
40 {  
    3 references  
41     public override void Sound()  
42     {  
43     }  
44 }  
45  
46 2 references  
47 class Cat : Animal  
48 {  
    3 references  
49     public override void Sound()  
50     {  
51         Console.WriteLine("Meo..Meo...");  
52     }  
53  
54 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

The screenshot shows a Visual Studio code editor with three tabs at the top: 'PetStore.cs*', 'InhDemo1.cs', and 'ConsoleApp*'. The 'InhDemo1.cs' tab is active, displaying the following C# code:

```
35 }  
36 }  
37 }  
38  
39     2 references  
40     sealed class GR : Dog  
41     {  
42         3 references  
43         public override void Sound()  
44         {  
45         }  
46  
47     0 references  
48     class GGR : GR  
49     {  
50     }  
51  
52     2 references  
53     class Cat : Animal  
54     {  
55         3 references  
56         public override void Sound()  
57     }
```

A tooltip is displayed over the inheritance line 'class GGR : GR'. The tooltip contains the following information:

- CS0509: 'GGR': cannot derive from sealed type 'GR'
- Show potential fixes (Alt+Enter or Ctrl+.)

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

```
2 references
class Cat : Animal
{
    3 references
    public override void Sound()
    {
        Console.WriteLine("Meo..Meo...");
    }

    3 references
    public override void Eat()
    {
        Console.WriteLine("Nog-veg");
    }
}
0 references
```

What are Abstract Classes ?

**Abstract classes may or may not have a Implementation
in our case abstract class has the implementation**

```
2 references
abstract class Dog : Animal
{
    3 references
    public override void Eat()
    {
        I
        Console.WriteLine("Veg and Non-veg");
    }
}
```

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

PetStore.cs X InhDemo1.cs ConsoleApp*

ConsoleApp

```
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      abstract class Animal
10     {
11         public void Walk()
12         {
13             Console.WriteLine("Use 4 legs to walk");
14         }
15
16         3 references
17         public abstract void Sound();
18         3 references
19         public abstract void Eat();
20
21         //public virtual void Sound()
22         //{
23         //    Console.WriteLine("Used to talk");
24         //}
25
26
27
28
29
2A
```

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

The screenshot shows a Visual Studio code editor window with the following code:

```
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      interface ICompanyRepo
10     {
11         void AddCompany();
12         void ListCompany();
13     }
14
15     interface IProductRepo
16     {
17         void AddProduct();
18         void ListProduct();
19     }
}
```

The code defines two interfaces: `ICompanyRepo` and `IProductRepo`. The `AddCompany()` method in `ICompanyRepo` is highlighted with a blue selection bar. The code editor interface at the bottom shows "121 %", a green checkmark icon, and the text "No issues found".

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

and **all 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 (Concerete 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 concerete class bv passing a parameter color as red

DOSA

(Abstract Class)

Explanation: When you go to hotel and ask for chapati, whether he ask you what kind of chaapati you want round or not, thin or not.....

No right

But when you ask for dosa, definitely 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 concrete

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