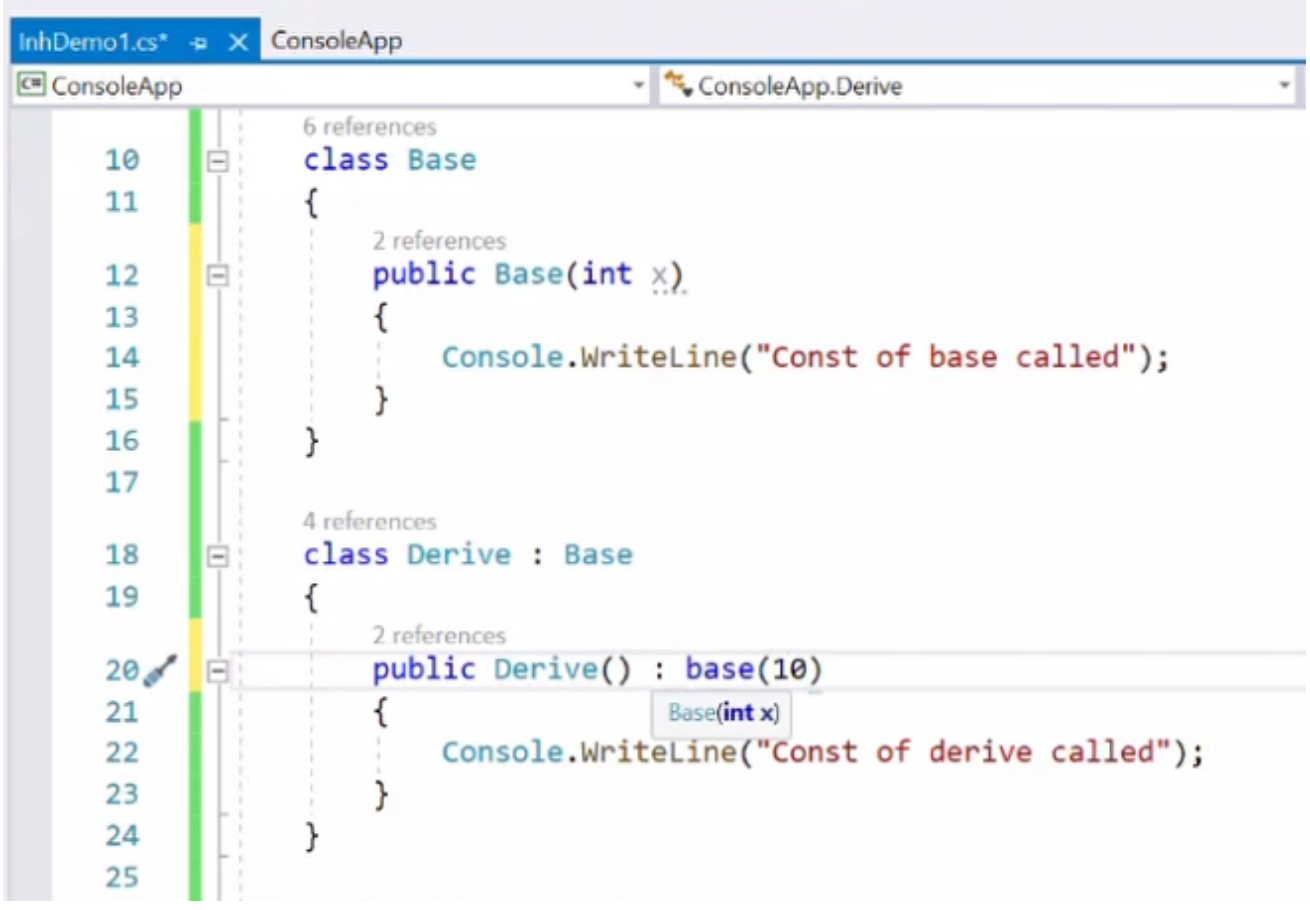
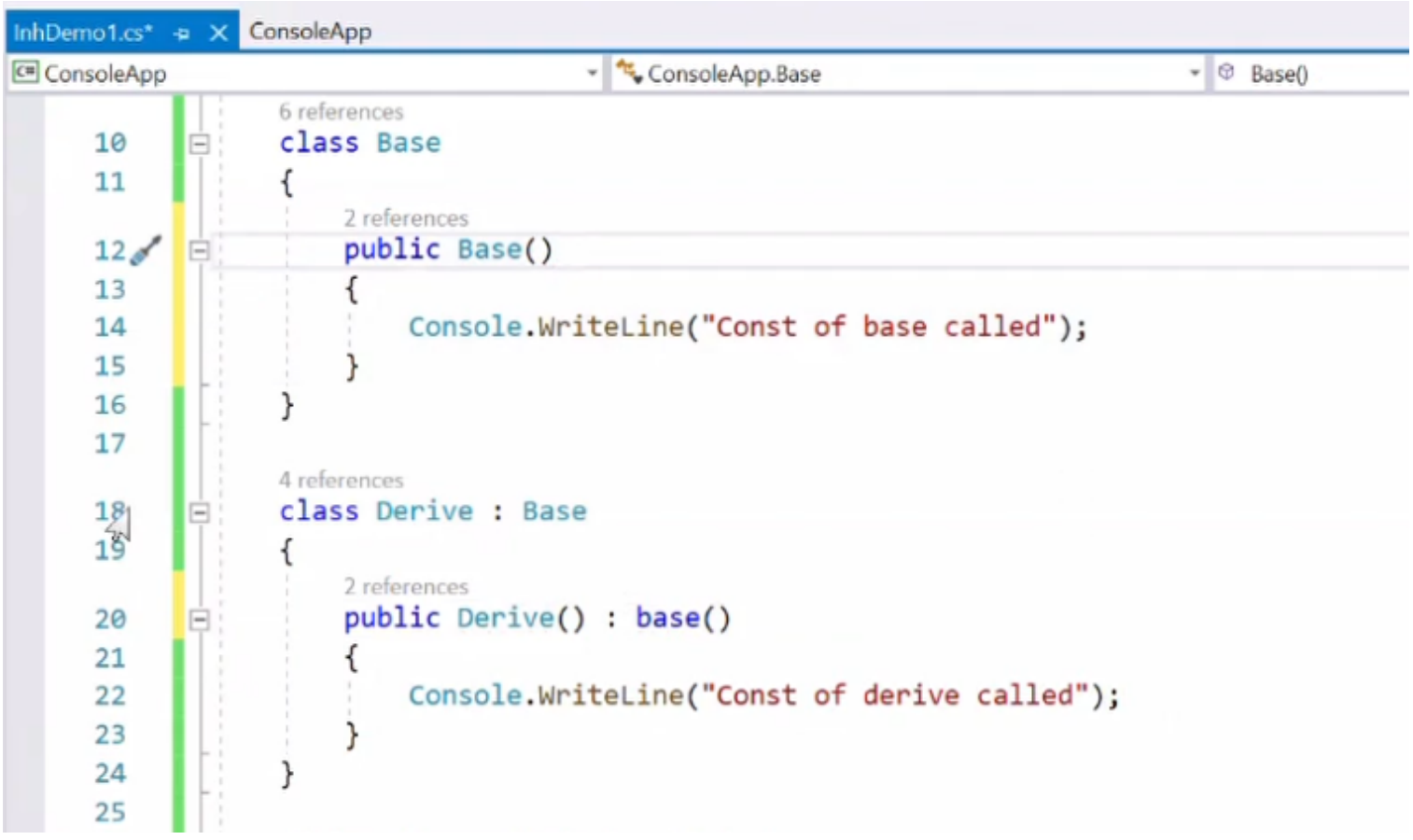
C# - 10



By default it will always have this signature

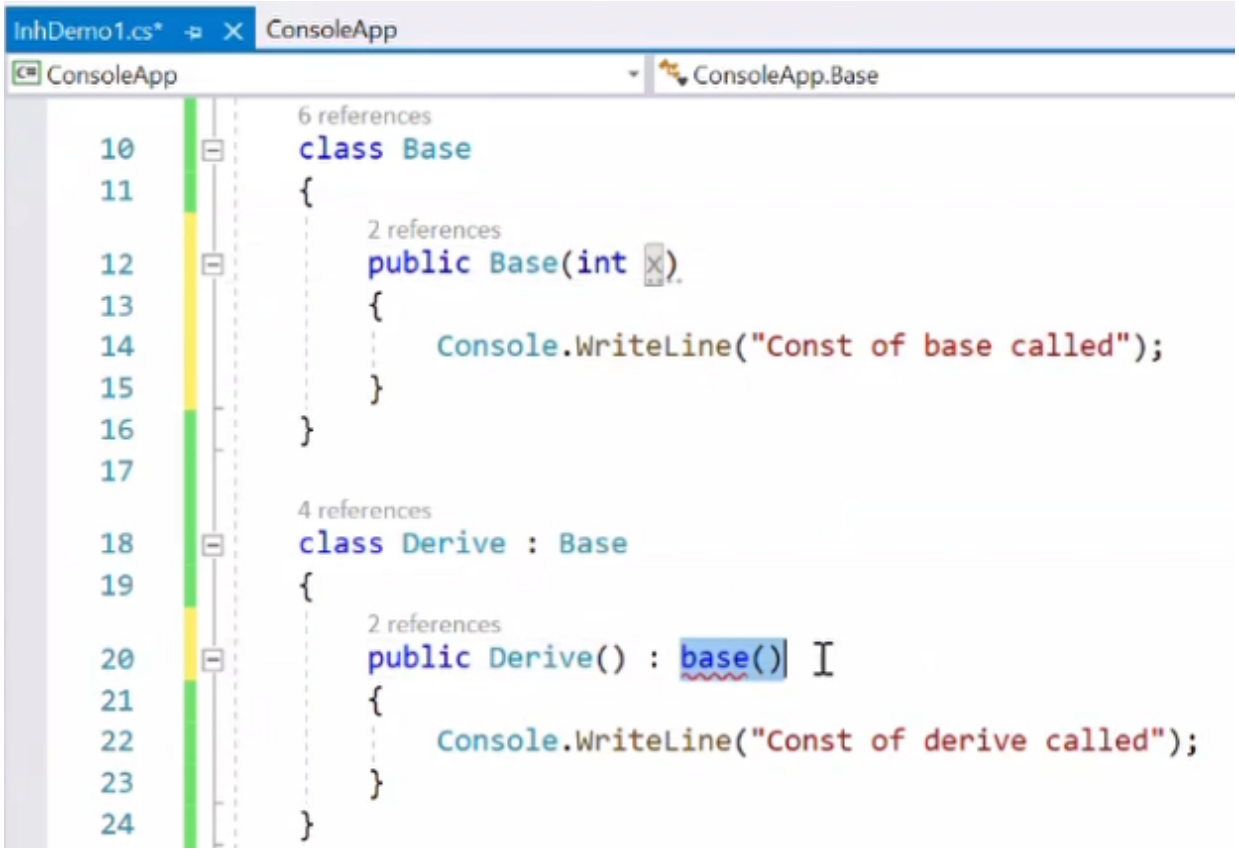


meaning your derived constructor will always have a : base()

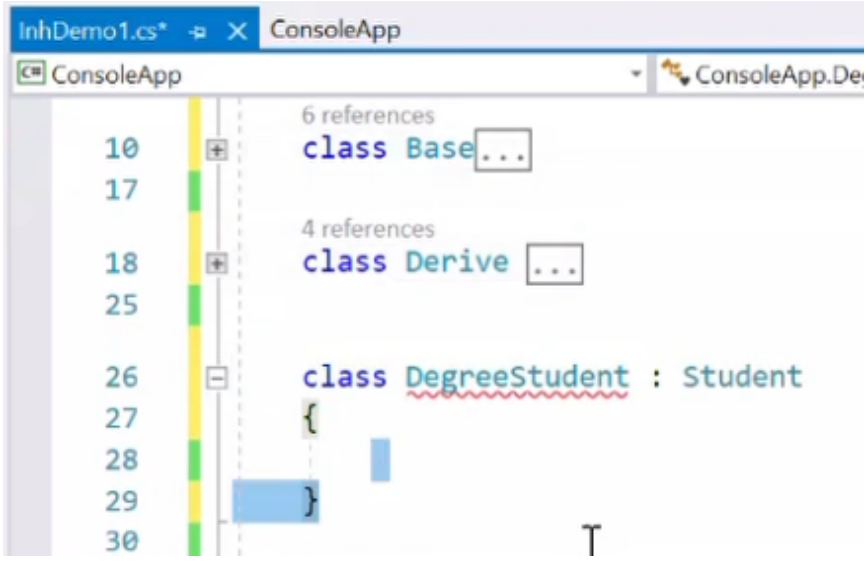
meaning it will always call the base class constructor, it will not do automatically

```
2 references
public Derive() : base()
{
```

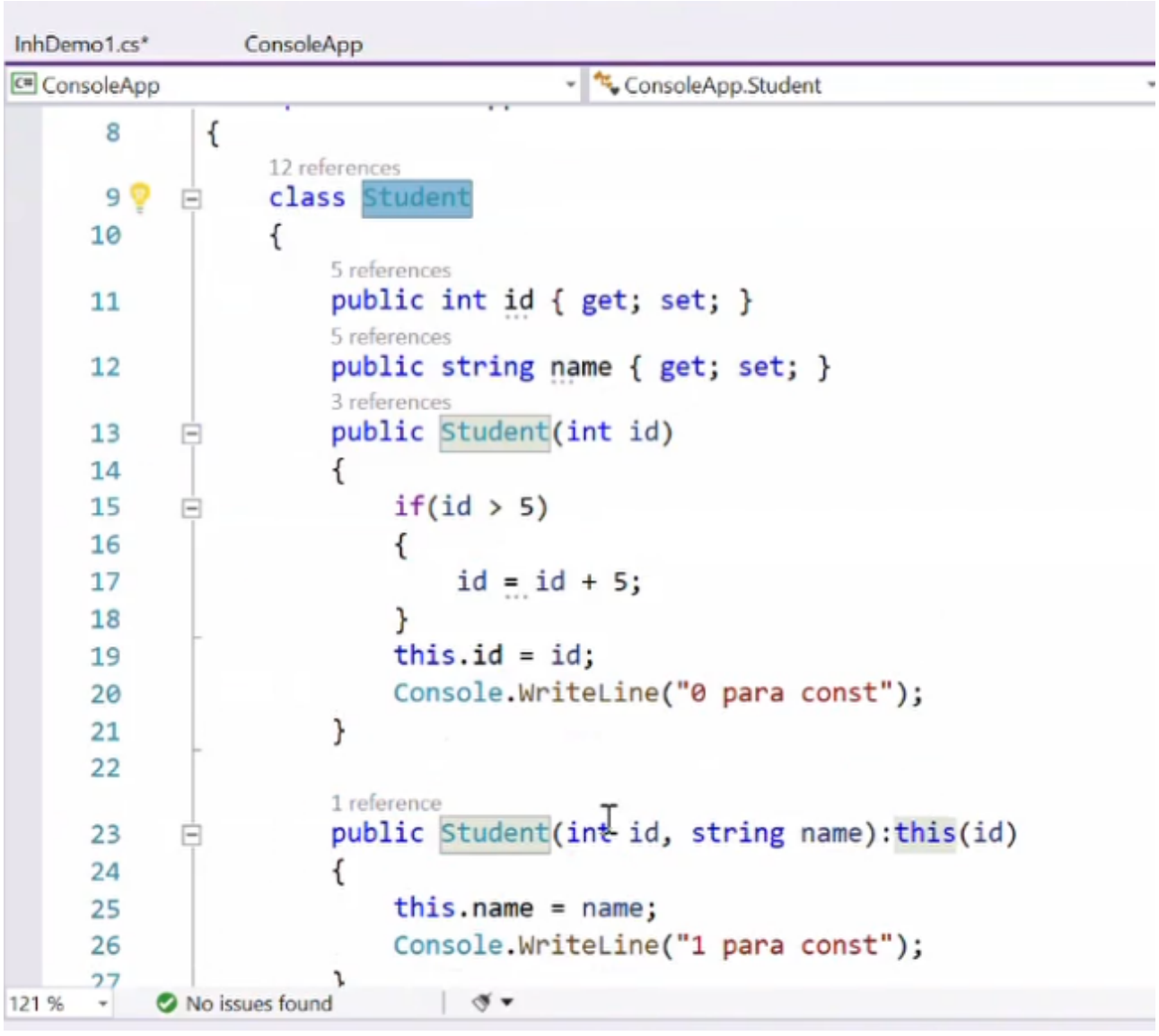
This piece of code written internally, so that's how I'm not bothered to write this code, but when I specify here int x



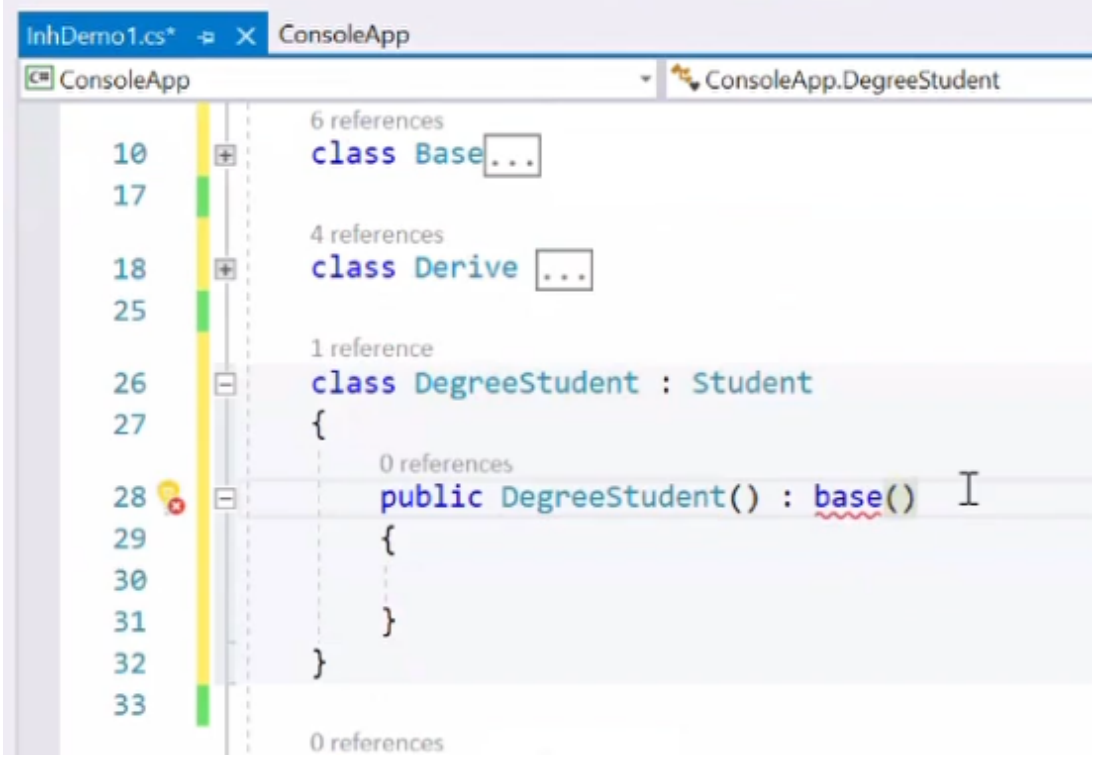
so this piece of code will not work, because I don't have any constructor which is not taking any parameter, So at this time you need to explicitly pass some value



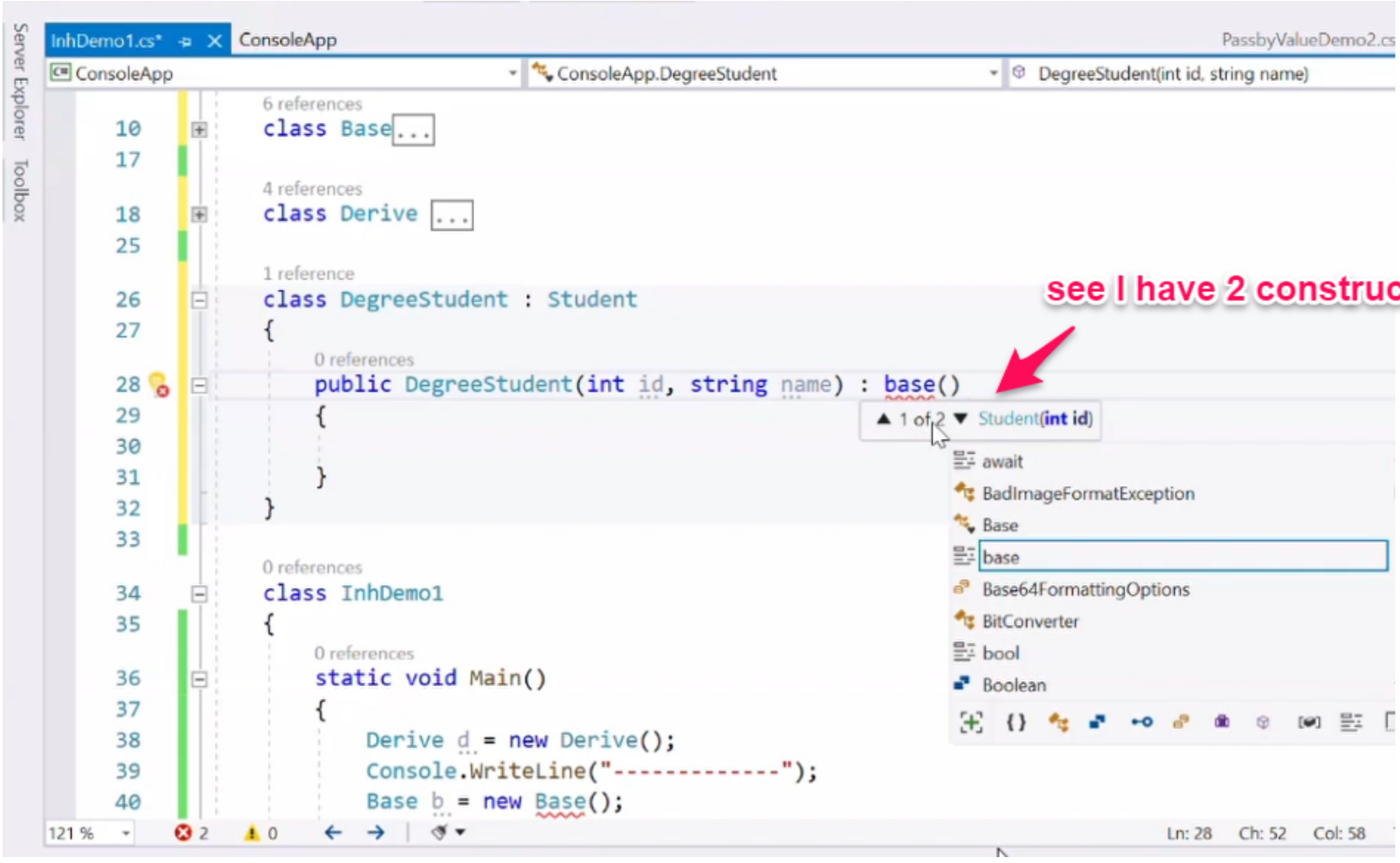
So here I'm getting Error because, my student dosen't support , 0 parameterised constructor

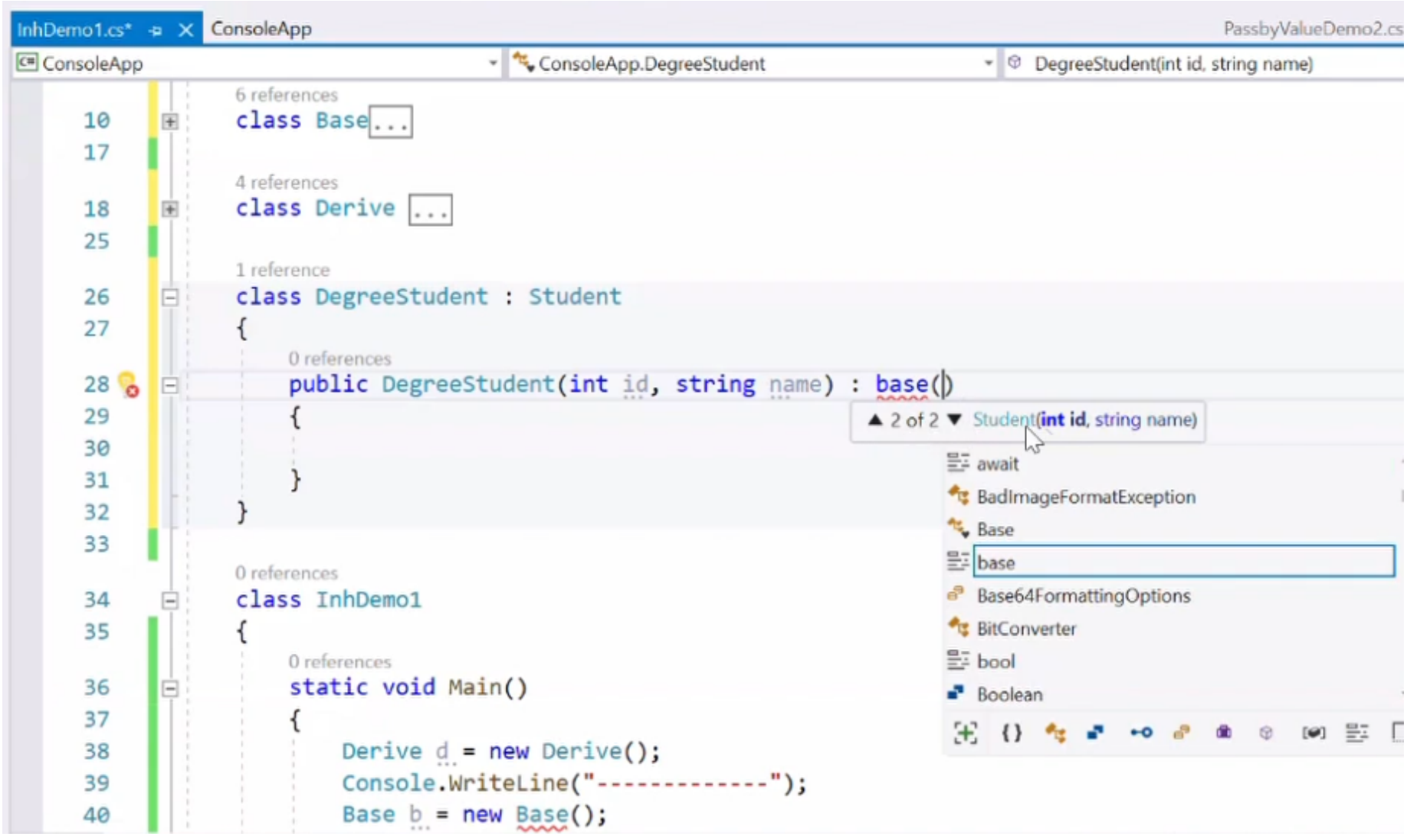

InhDemo1.cs*
ConsoleApp
ConsoleApp
ConsoleApp.Student
{
12 references
class Student
{
5 references
public int id { get; set; }
5 references
public string name { get; set; }
3 references
public Student(int id)
{
if(id > 5)
{
id = id + 5;
}
this.id = id;
Console.WriteLine("0 para const");
}
1 reference
public Student(int id, string name):this(id)
{
this.name = name;
Console.WriteLine("1 para const");
}
121 %
No issues found

So internally, Degree will have a constructor which will in return call the 0 parameterized constructor of the base class of the student class which we don't have



so base class is expecting 1 or 2 parameter based on this


InhDemo1.cs*
ConsoleApp
PassbyValueDemo2.cs
ConsoleApp
ConsoleApp.DegreeStudent
DegreeStudent(int id, string name)
Server Explorer
Toolbox
6 references
10 class Base...
17
4 references
18 class Derive ...
25
1 reference
26 class DegreeStudent : Student
27 {
0 references
28 public DegreeStudent(int id, string name) : base()
29 {
30
31 }
32 }
33
0 references
34 class InhDemo1
35 {
0 references
36 static void Main()
37 {
38 Derive d = new Derive();
39 Console.WriteLine("-------------");
40 Base b = new Base();
see I have 2 constructor here
1 of 2
Student(int id)
await
BadImageFormatException
Base
base
Base64FormattingOptions
BitConverter
bool
Boolean
121 %
2
0
Ln: 28
Ch: 52
Col: 58

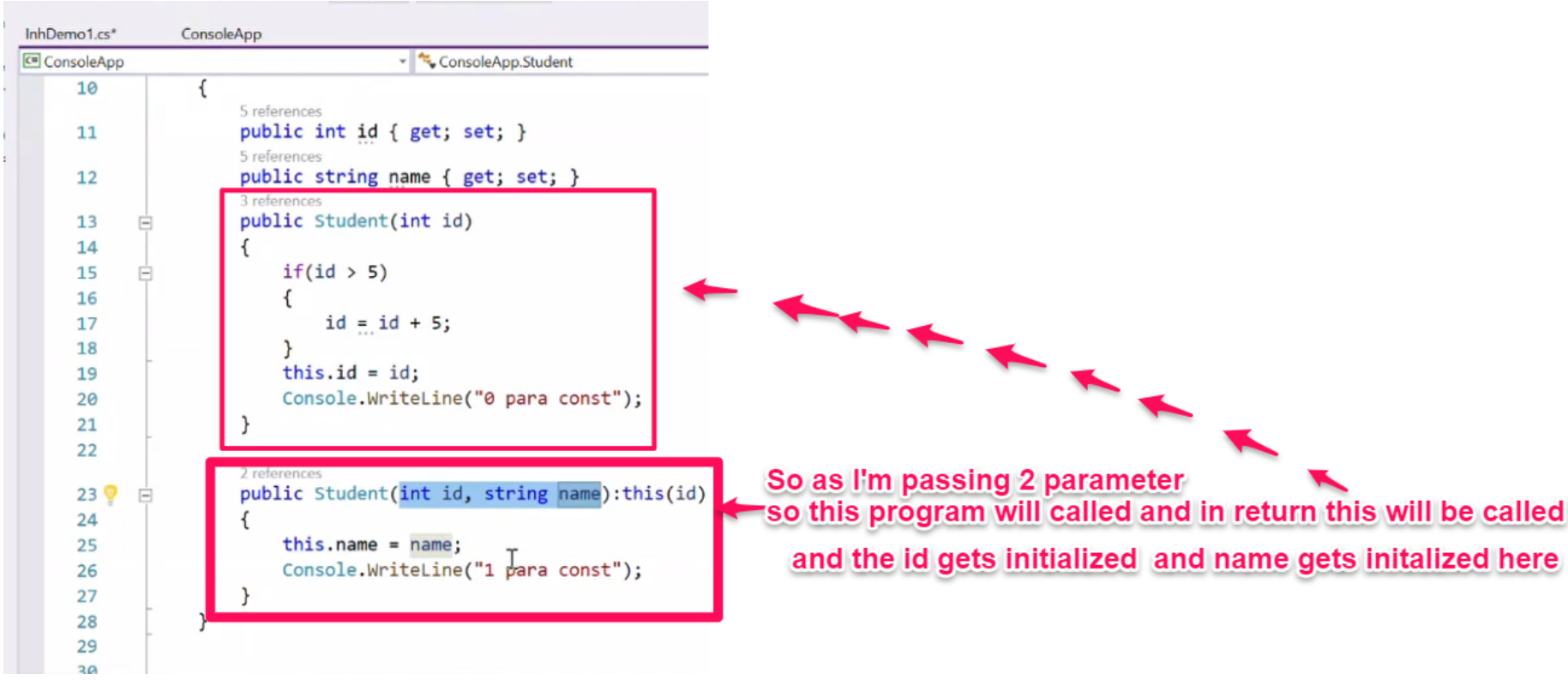


This is called Overloading

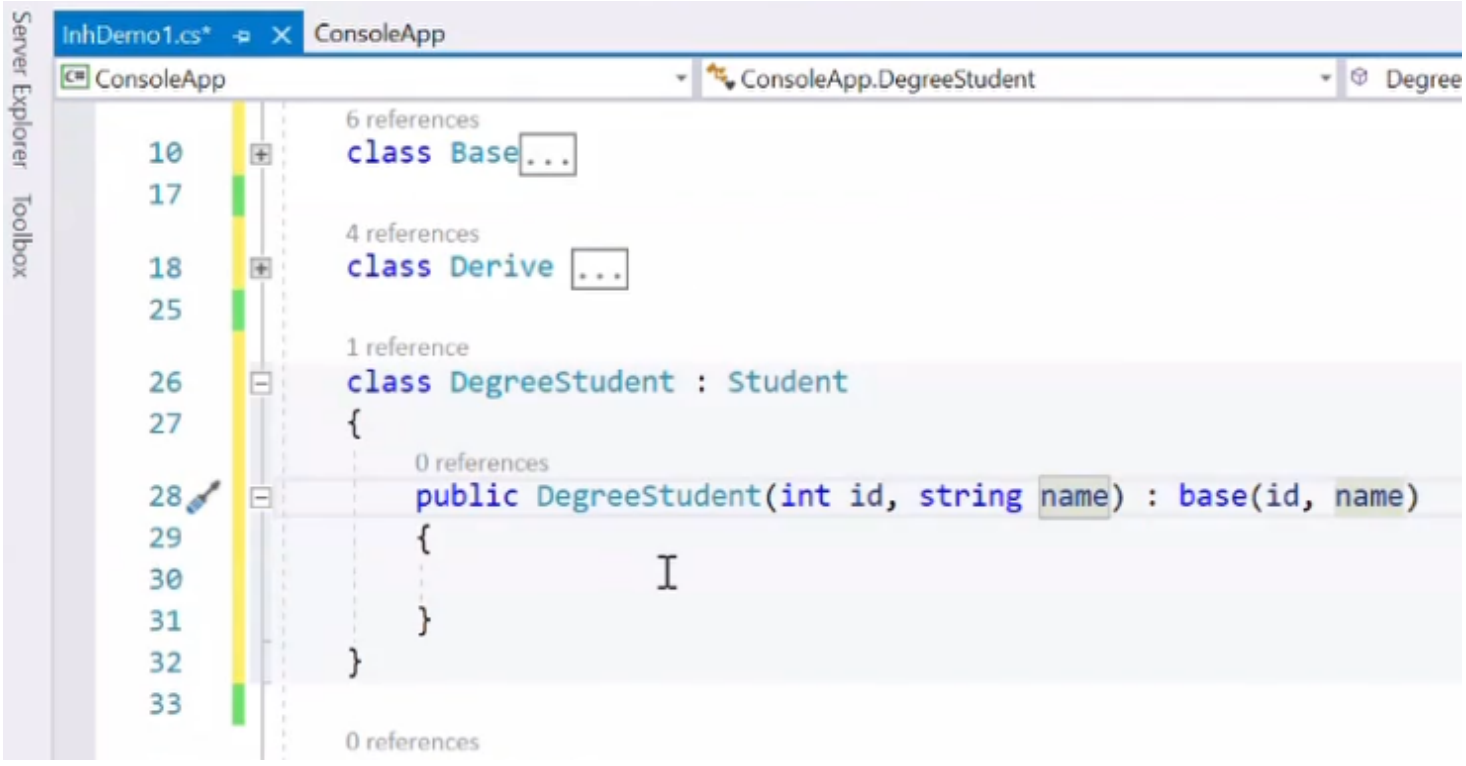
Look at this at the compile time I get to know which method will be called this is Static Polymorphism

at the compile time I get to know that's why we called it as Static, Static means I know the behaviour

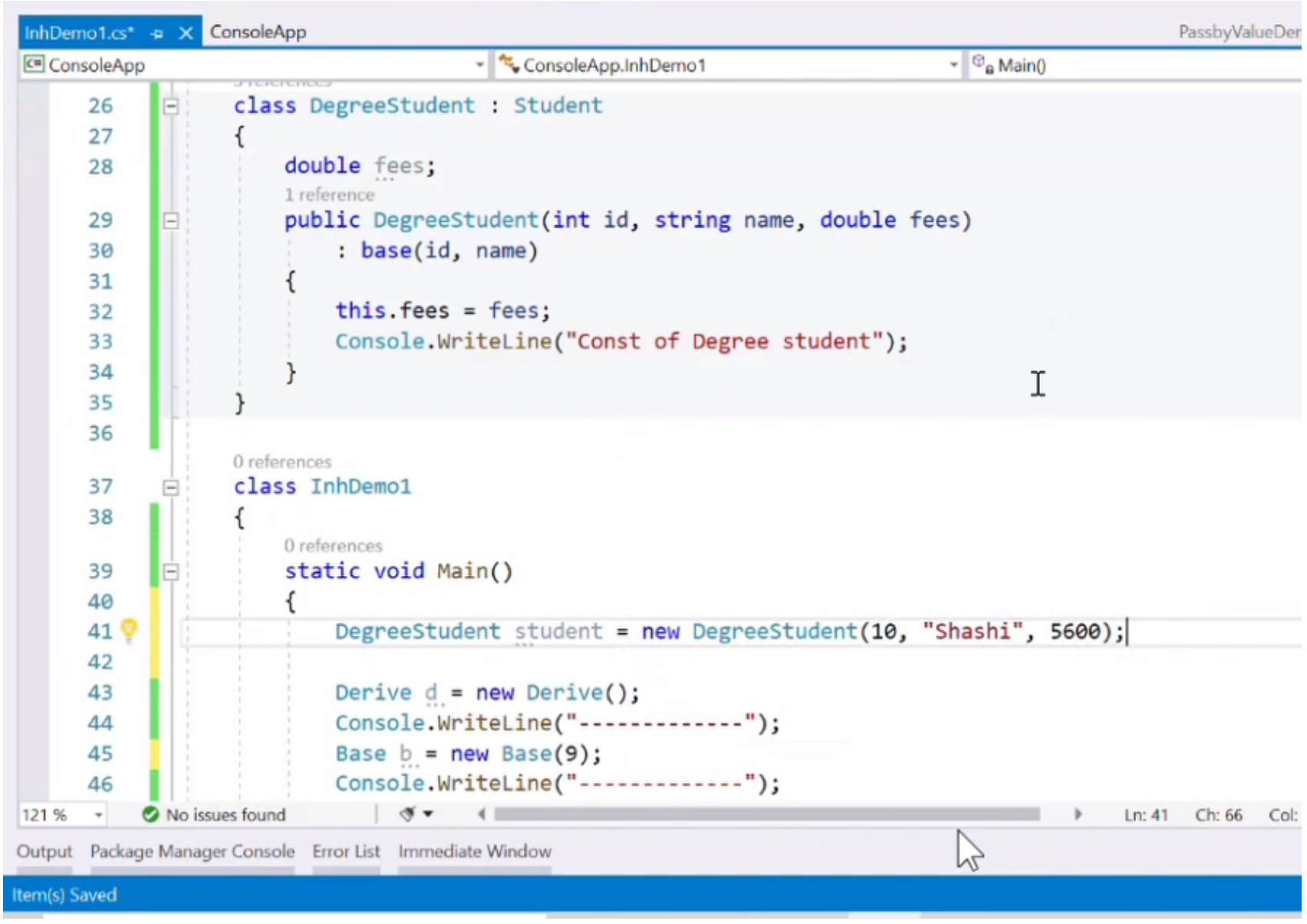
So it can have the **id** and **name** or only the **id**



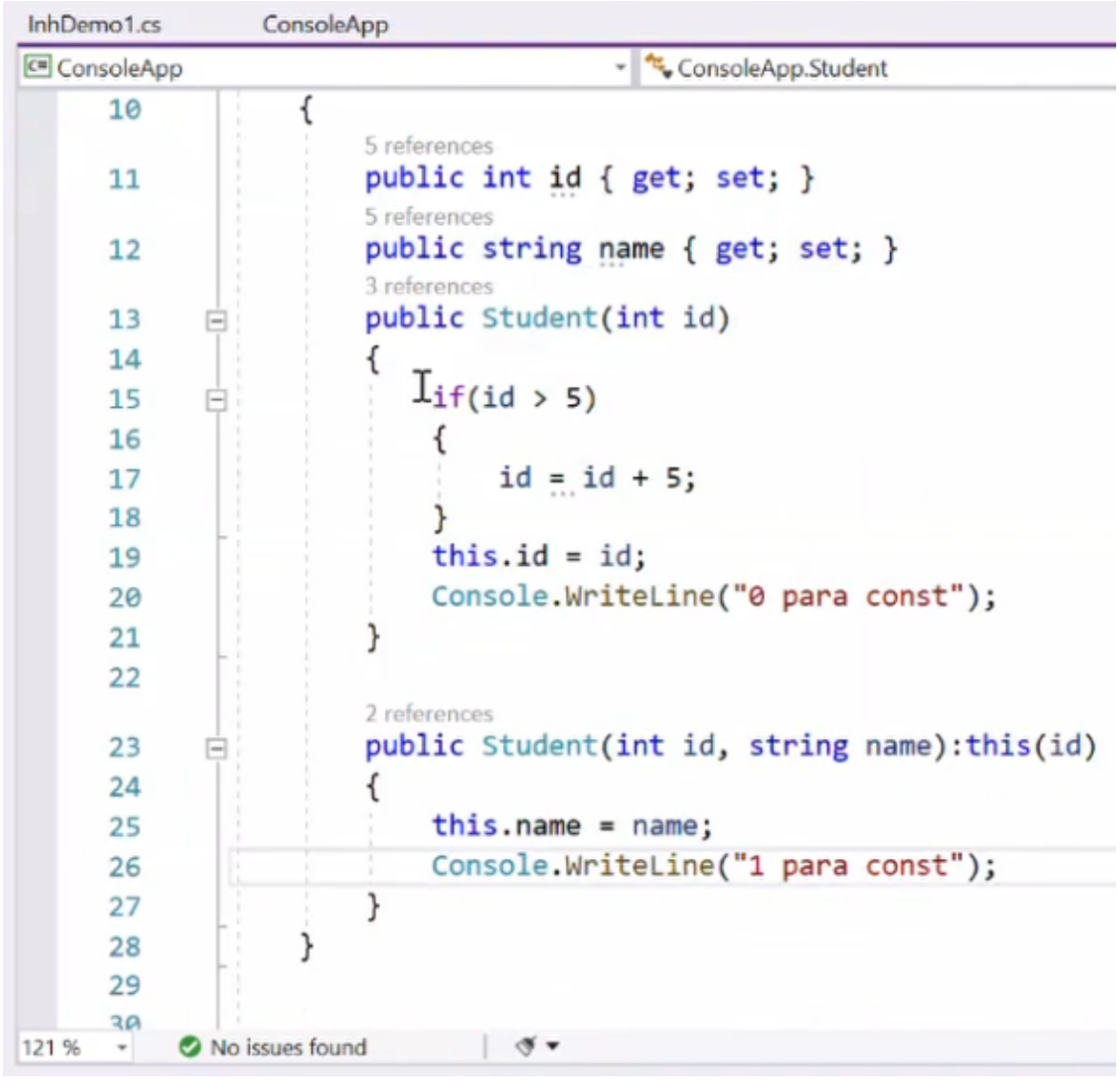
then it will come back to this program



Suppose I have a double fee and

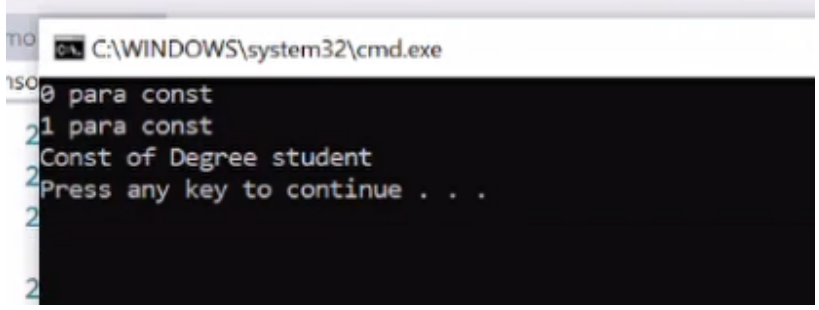

InhDemo1.cs*
ConsoleApp
PassbyValueDer
ConsoleApp
ConsoleApp.InhDemo1
Main()
class DegreeStudent : Student
{
double fees;
1 reference
public DegreeStudent(int id, string name, double fees)
: base(id, name)
{
this.fees = fees;
Console.WriteLine("Const of Degree student");
}
}
0 references
class InhDemo1
{
0 references
static void Main()
{
DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
Derive d = new Derive();
Console.WriteLine("-------------");
Base b = new Base(9);
Console.WriteLine("-------------");
121 %
No issues found
Ln: 41
Ch: 66
Col:
Output
Package Manager Console
Error List
Immediate Window
Item(s) Saved

What will be the output of this
So here we are passing 10 shashi and 5600
but it inherit from base
so base will be called with 2 parameter 10 and shashi



so it will go here which is taking 2 parameter but this is also calling the constructor with 1 parameter

so 1st it will print **0 para const**
then control go to its child which print **1 para const**
then control go to its child which print **Const of Degree student**



So In **Polymorphism**

we have 1st **Static Polymorphism (Over Loading)** where i know which method gets call that's why we call it as **Compile time binding/ Early Binding** also

----------**Here we have** ----------
----------> Constructor Over Loading
----------> Method Over Loading
----------> Operator Over Loading
=====**In Operator Over Loading we have**
========> Unary Over Loading
========> Binary Over Loading

2nd We have **Dynamic Polymorphism (Over Riding)**

What is a difference between a Constructor and a Method ?

Constructors will be called only once, When you are creating object it gets called

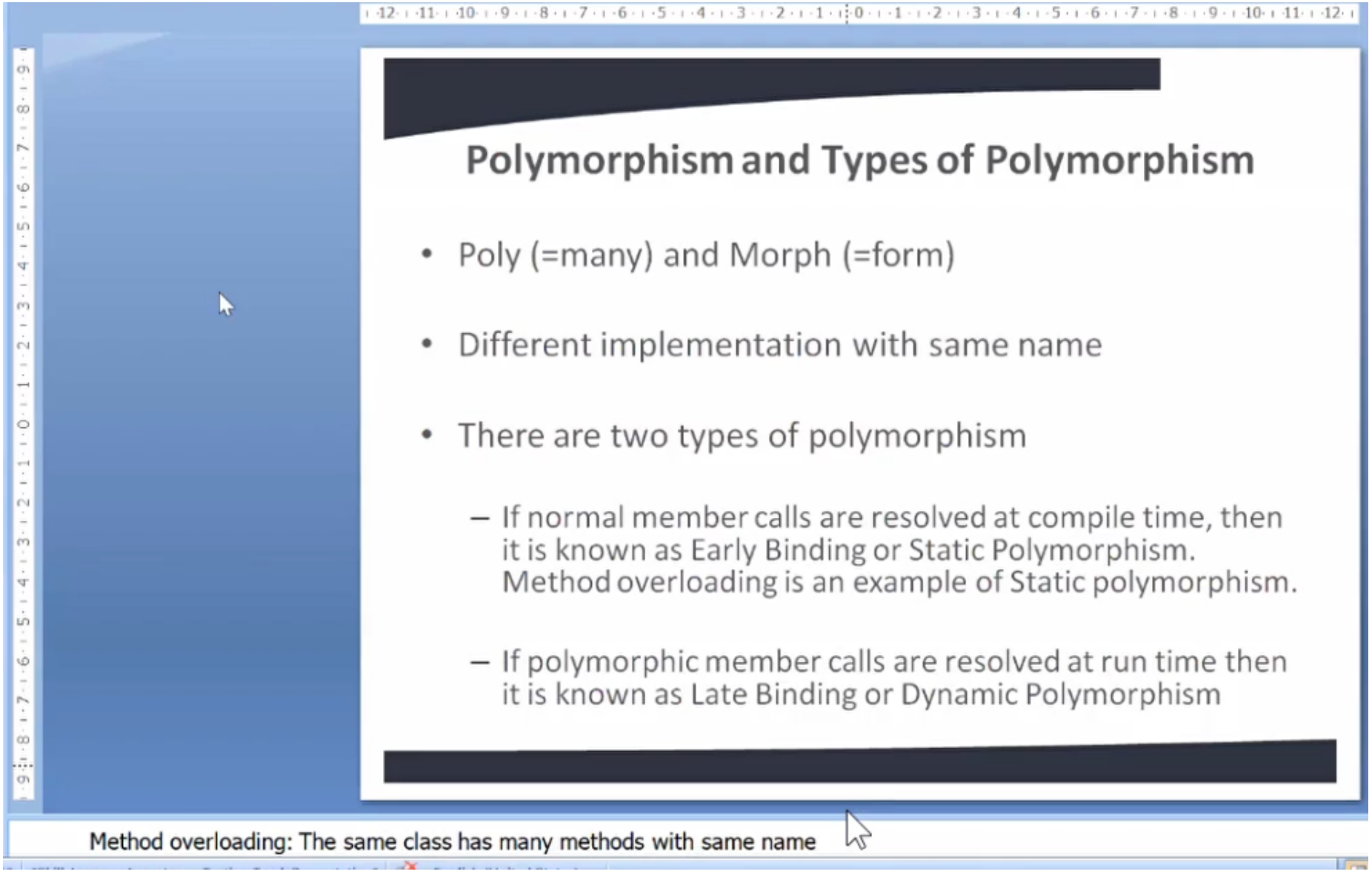
So when you want to do some kind of initialization of the data member, we'll use the constructor

Methods can be called any time, you create a object and after creating a object, you can keep calling that method any number of time

Suppose Instead of two hand I want baby to have 4 hand, so where should i do it, In the constructor or in the method?

--> It Should be done at the constructor level

because after baby got born, then you cannot attach 2 hands ☺


Polymorphism and Types of Polymorphism
• Poly (=many) and Morph (=form)
• Different implementation with same name
• There are two types of polymorphism
– If normal member calls are resolved at compile time, then it is known as Early Binding or Static Polymorphism. Method overloading is an example of Static polymorphism.
– If polymorphic member calls are resolved at run time then it is known as Late Binding or Dynamic Polymorphism
Method overloading: The same class has many methods with same name

Method Overloading

- It is an example of early binding or static polymorphism
- Which method will be called that is decided during compile time
- All the overloaded methods, with same name, are part of single class
- Method signatures must be unique within a class
- Signature definition makes the difference

Criteria for Signature Difference

- No of Parameters
- Data Type of Parameters
- Position of Parameters

Which Does Not Have An Impact On Signature?

- Name of parameter
- Return type of method

Declaring Overloaded Methods

```
class Calculation
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }
    public double Add(double a, double b, int c)
    {
        return a + b + c;
    }
    public double Add(double a, int b, double c)
    {
        return a + b + c;
    }
}
```

Calculation is having four overloaded 'Add' methods with four different signatures

```
class Program
{
    static void Main(string[] args)
    {
        Calculation calcobject = new Calculation();
        calcobject.Add(12, 13);
        calcobject.Add(12, 13, 14);
        calcobject.Add(12.45, 13.34, 14);
        calcobject.Add(12.45, 13, 14.56);
    }
}
```

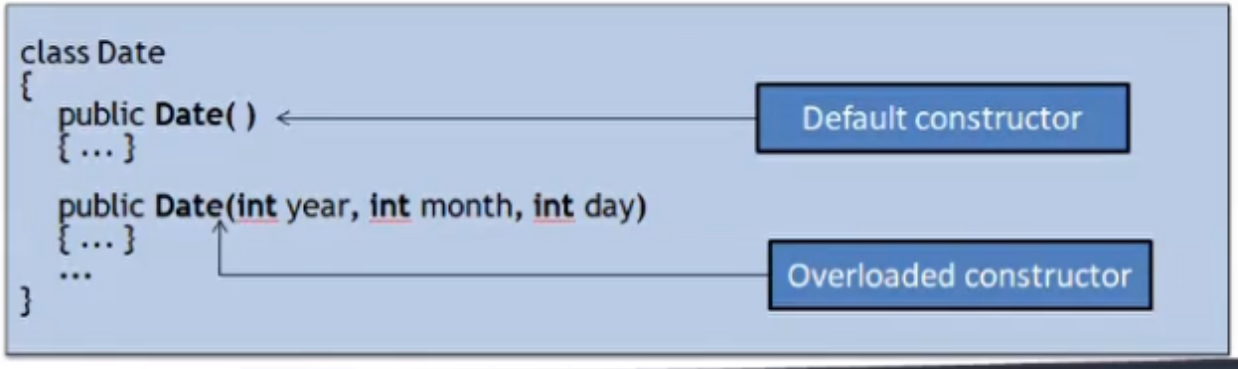
During method calls, each and every method call will be attached to proper overloaded method with the help of method signature

When to Use and When Not to

- Consider using overloaded methods when:
  - You have similar methods that require different parameters
  - You want to add new functionality to existing code
- Do not overuse because:
  - Hard to debug
  - Hard to maintain

Constructor Overloading

- Constructors are methods and hence can be overloaded
  - Same scope, same name, different parameters
  - Allows objects to be initialized in different ways
- WARNING
  - If you write a overloaded constructor for a class, the compiler does not create a default constructor



Example: Constructor Overloading

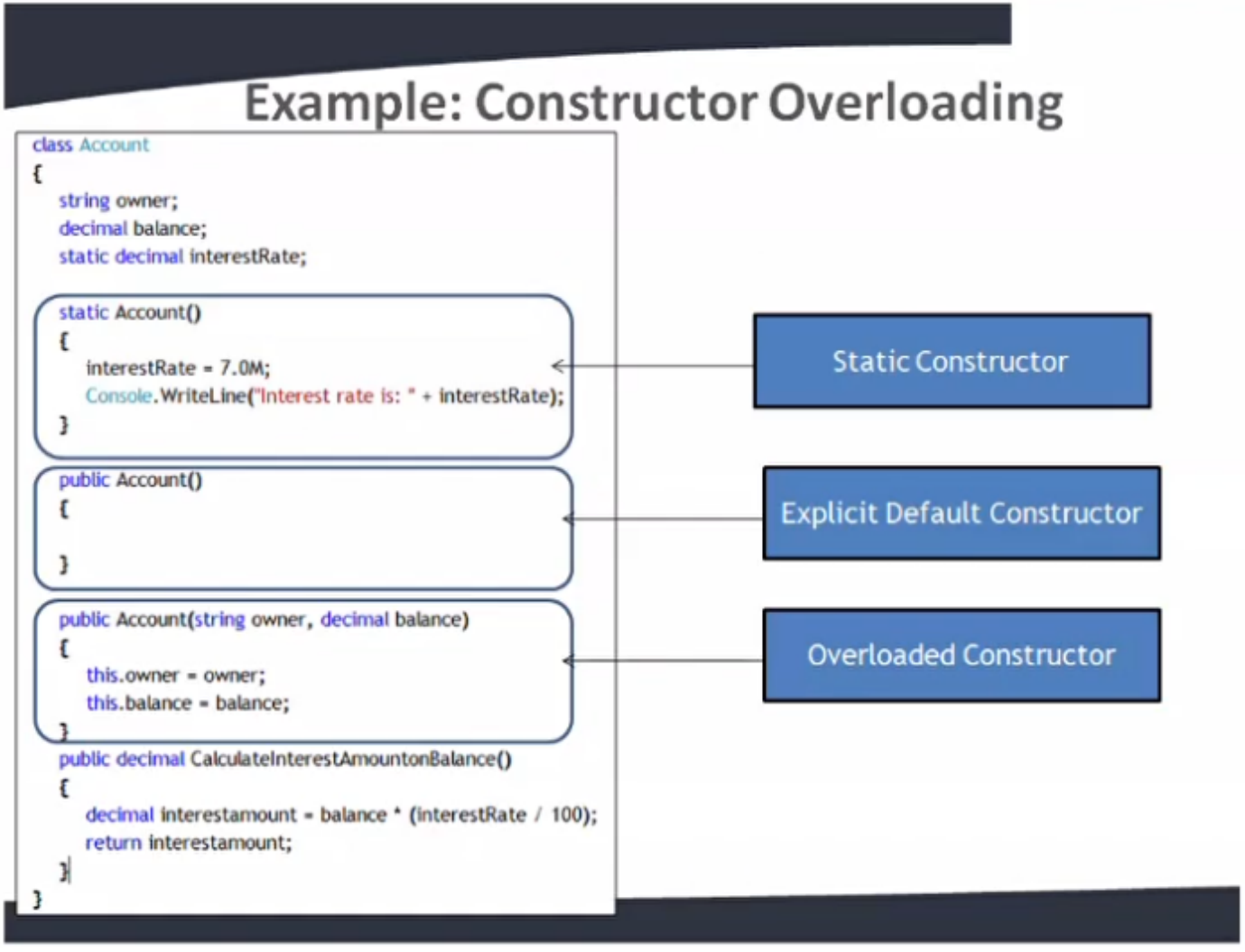
```
class Account
{
    string owner;
    decimal balance;
    static decimal interestRate;

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

    public Account()
    {

    }

    public Account(string owner, decimal balance)
    {
        this.owner = owner;
        this.balance = balance;
    }
    public decimal CalculateInterestAmountonBalance()
    {
        decimal interestamount = balance * (interestRate / 100);
        return interestamount;
    }
}
```



Example: Constructor Overloading

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

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

        Console.ReadLine();
    }
}
```

Calling Overloaded Constructor

‘base’ Keyword

- ‘base’ keyword is used to call any public or protected or internal or protected internal member of base class from the immediate derived class
- It can be used to call even any overloaded constructor of base class explicitly
- Can’t be used to invoke any private member of base class
- Can be used only from immediate child class of any base class, not from any child class present in the hierarchy in a multilevel inheritance

Calling Base Class Constructor Using base Keyword

- Whenever a derived class constructor gets called, always default constructor of base class gets called
- You can call any base class constructor from child class explicitly, instead of default base class constructor
- Constructor declaration in child/derived class must use the base keyword to call particular constructor of base class
- A private base class constructor cannot be accessed by a derived class
- Use the base keyword to qualify identifier scope

```
class Token
{
    protected Token(string name) { ... }
    ...
}
class CommentToken: Token
{
    public CommentToken(string name) : base(name) { }
    ...
}
```

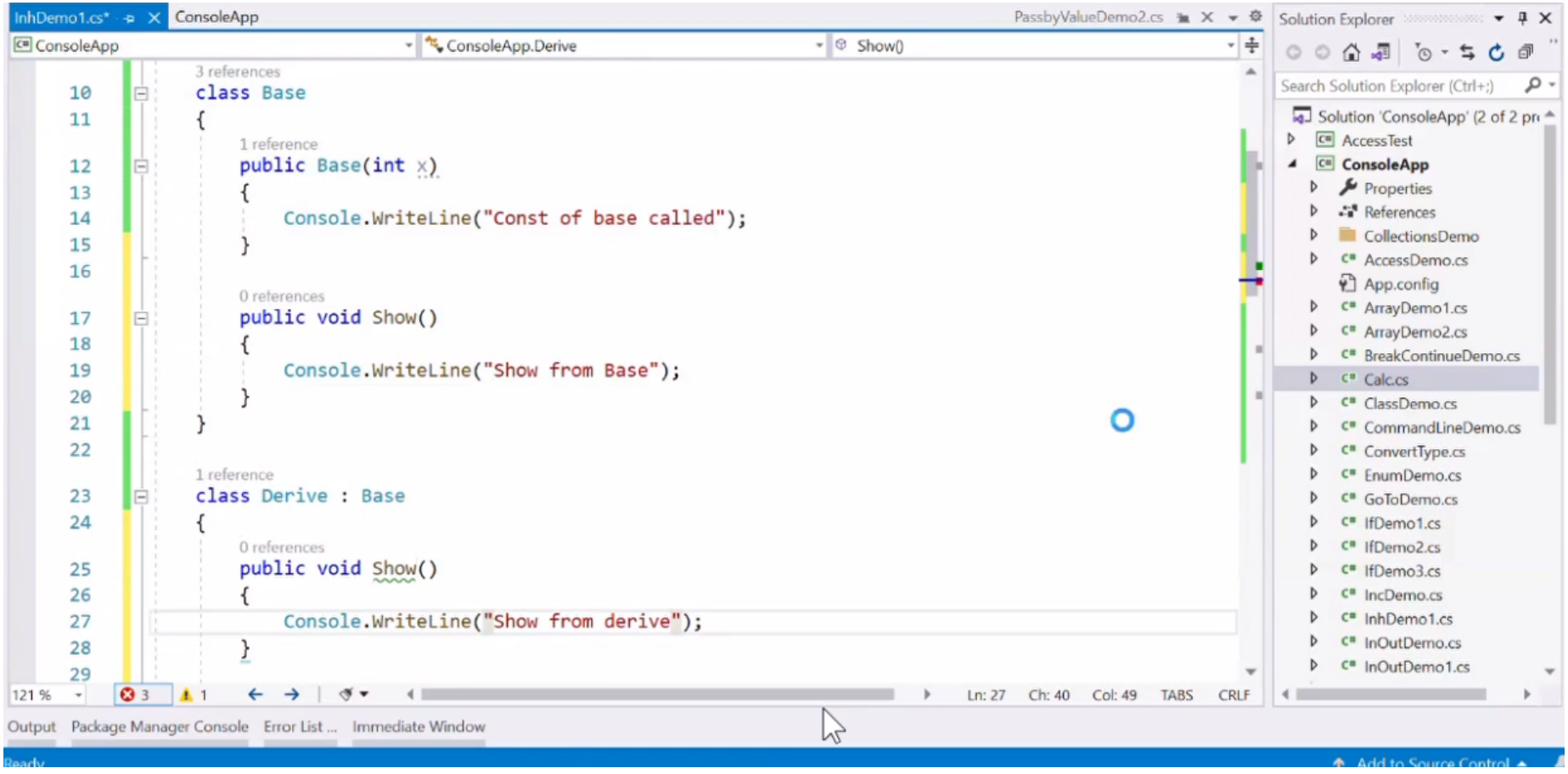
Calling Base Class Method Using 'base' Keyword

- Even any method or any other member (fields, properties) that are not private can be called from immediate child class using 'base' keyword

- You can call only immediate base class method from the child class

we can also use base in

Suppose I have show method in both base and derived class



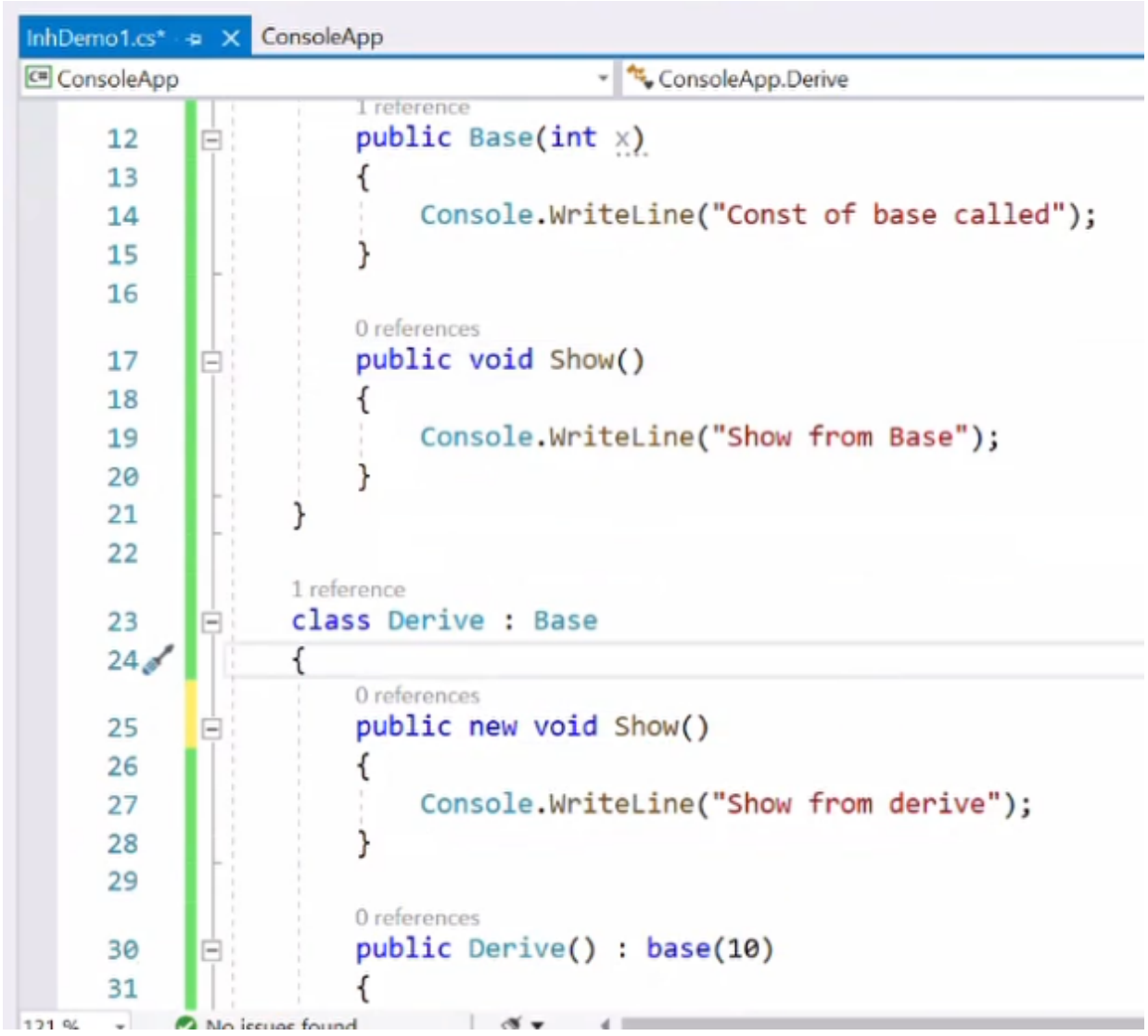
Now I don't want that base class show method override the derived class's show method

if i want to override then I make it virtual

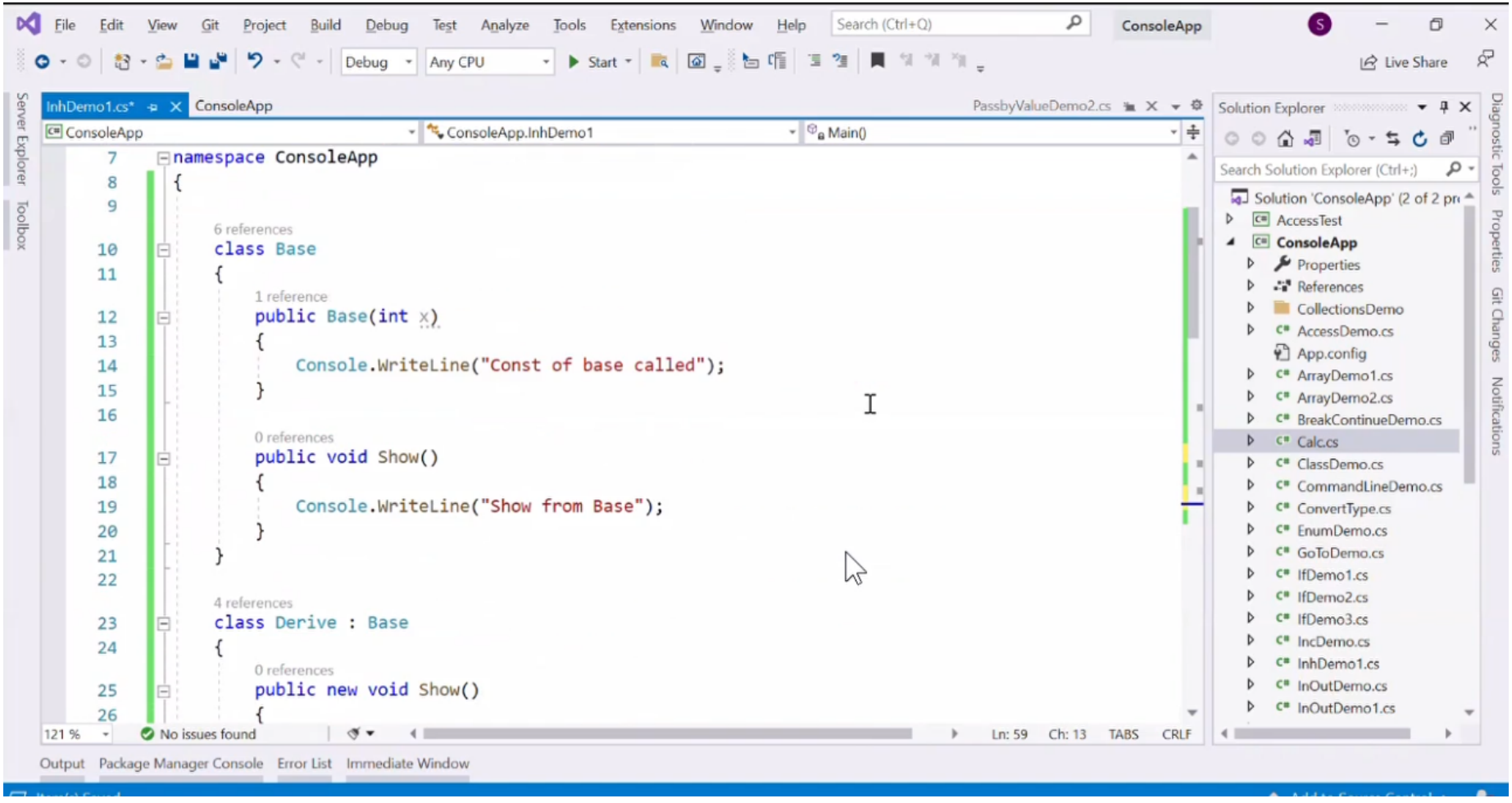
you can see here the green line it show warning

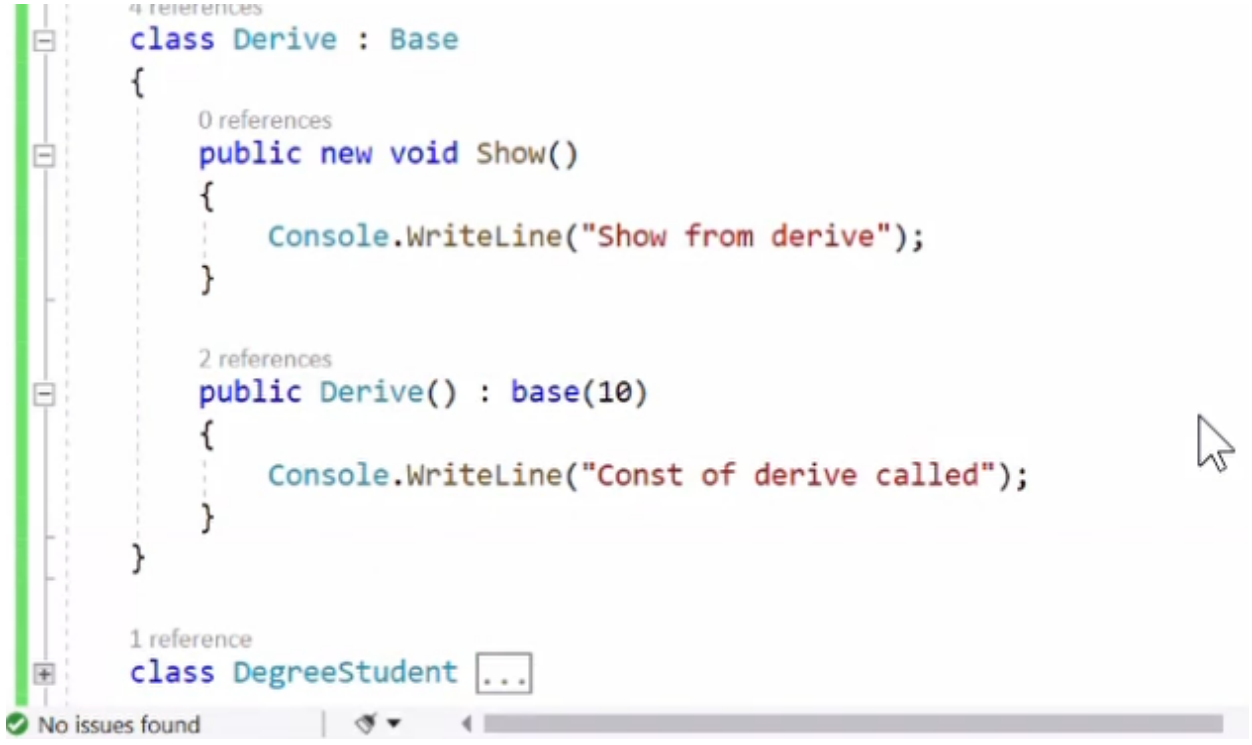
it says boss you already have the show method so you just hide this show

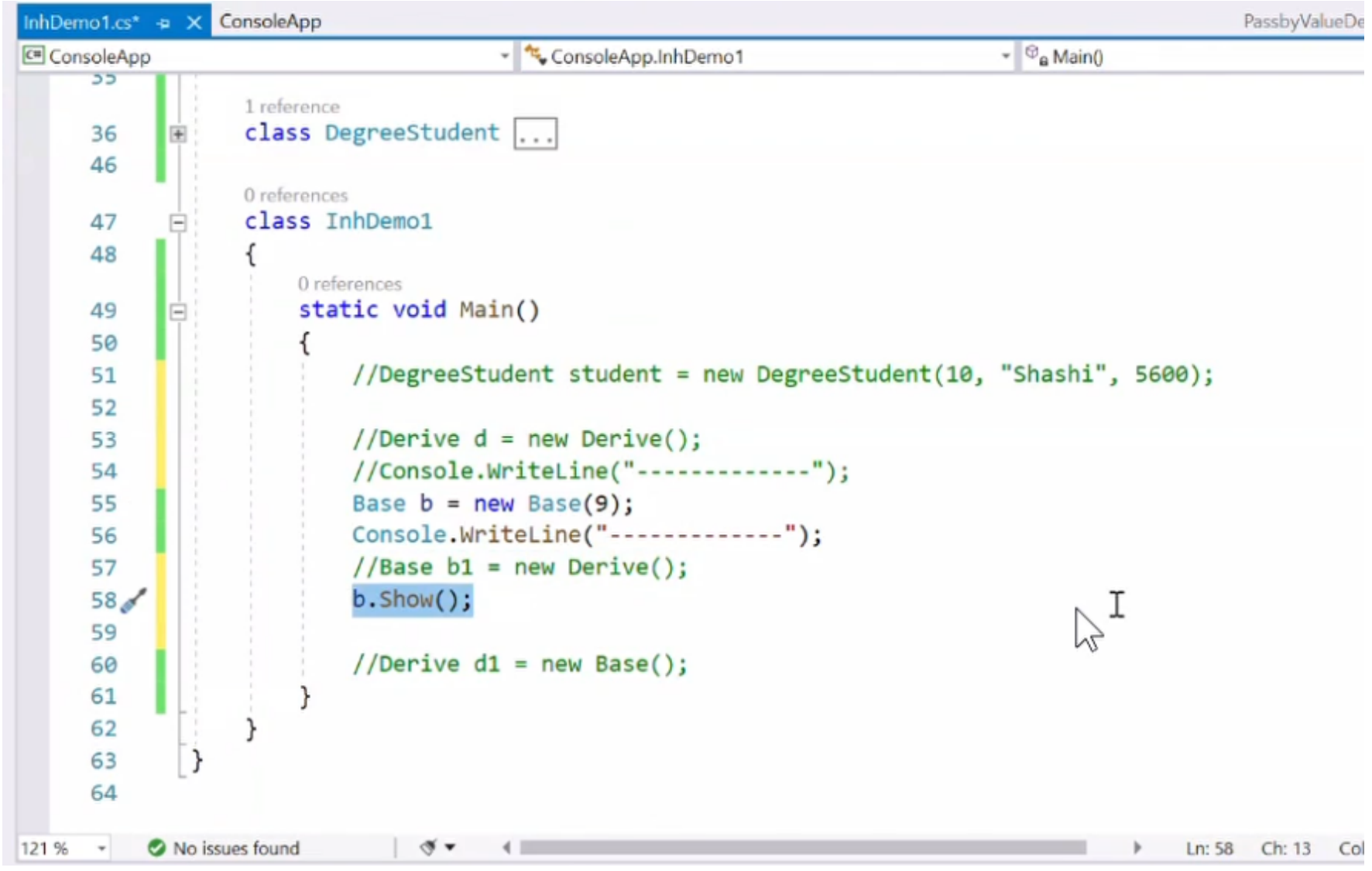
so to hide just say **new** here



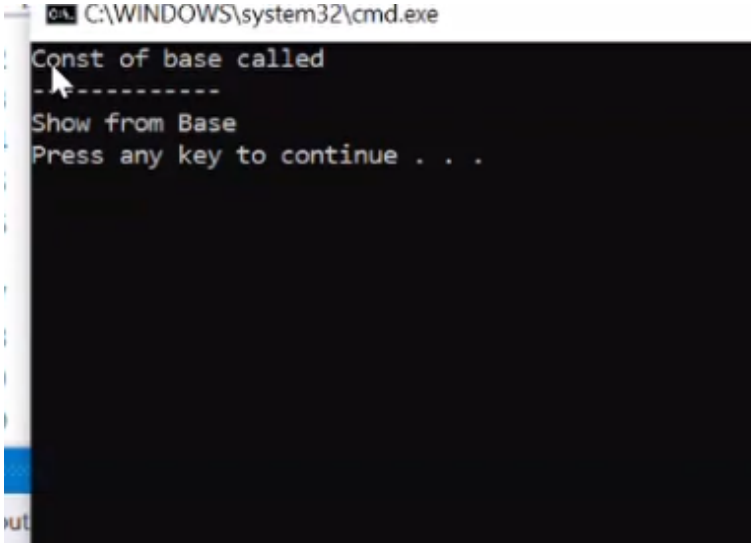
now when i create instance of base [b.show]()


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
ConsoleApp
Debug
Any CPU
Start
Live Share
InhDemo1.cs*
ConsoleApp
PassbyValueDemo2.cs
ConsoleApp
ConsoleApp.InhDemo1
Main()
namespace ConsoleApp
{
6 references
class Base
{
1 reference
public Base(int x)
{
Console.WriteLine("Const of base called");
}
0 references
public void Show()
{
Console.WriteLine("Show from Base");
}
}
4 references
class Derive : Base
{
0 references
public new void Show()
{
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
121 %
No issues found
Ln: 59
Ch: 13
TABS
CRLF
Output
Package Manager Console
Error List
Immediate Window
Server Explorer
Toolbox
Diagnostic Tools
Properties
Git Changes
Notifications

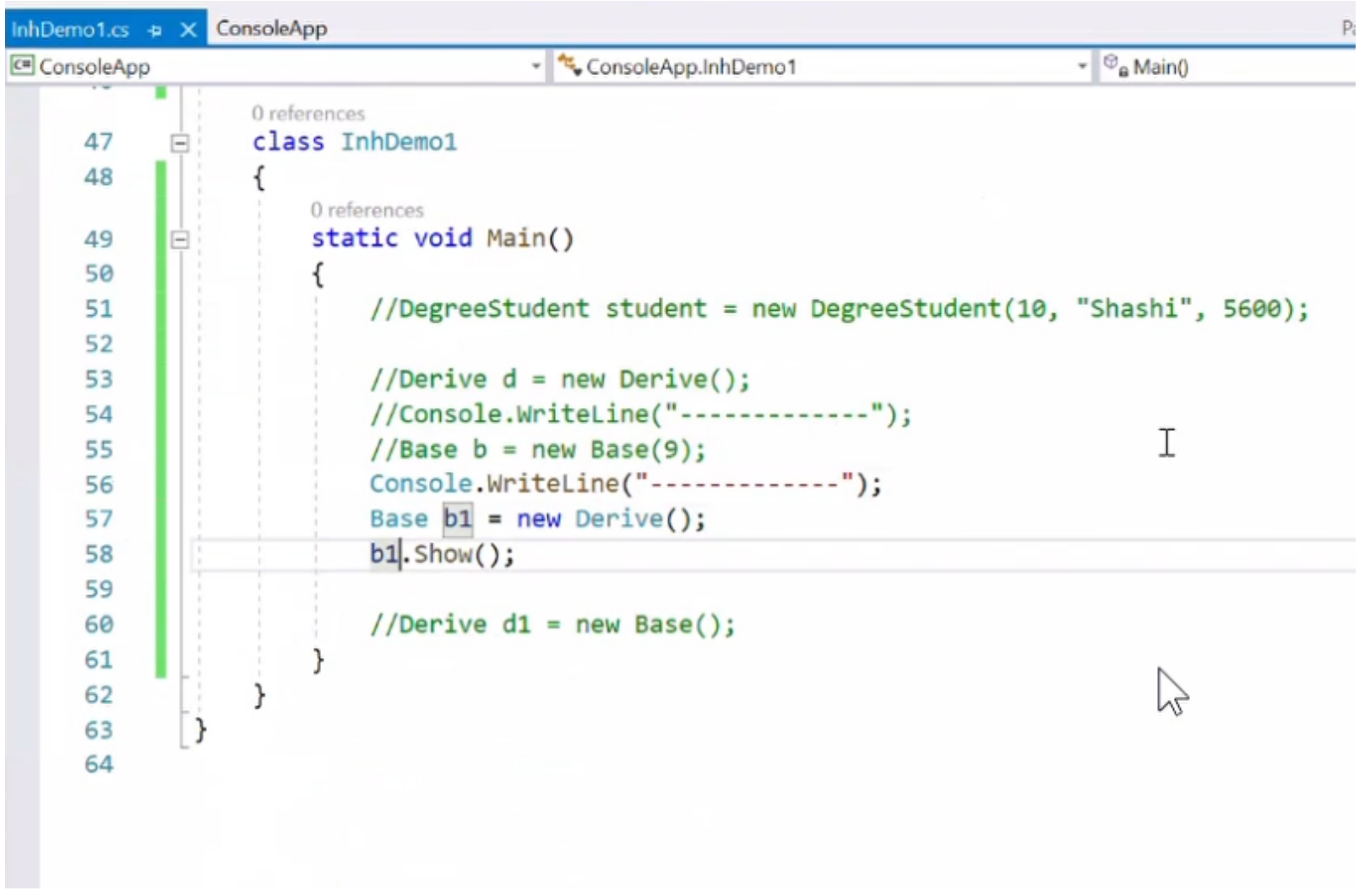

4 references
class Derive : Base
{
0 references
public new void Show()
{
Console.WriteLine("Show from derive");
}
2 references
public Derive() : base(10)
{
Console.WriteLine("Const of derive called");
}
}
1 reference
class DegreeStudent ...
No issues found

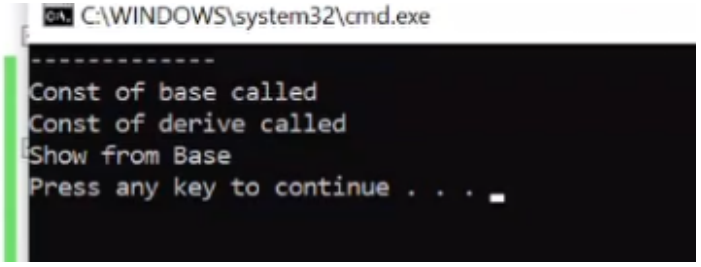


what will be the output of this

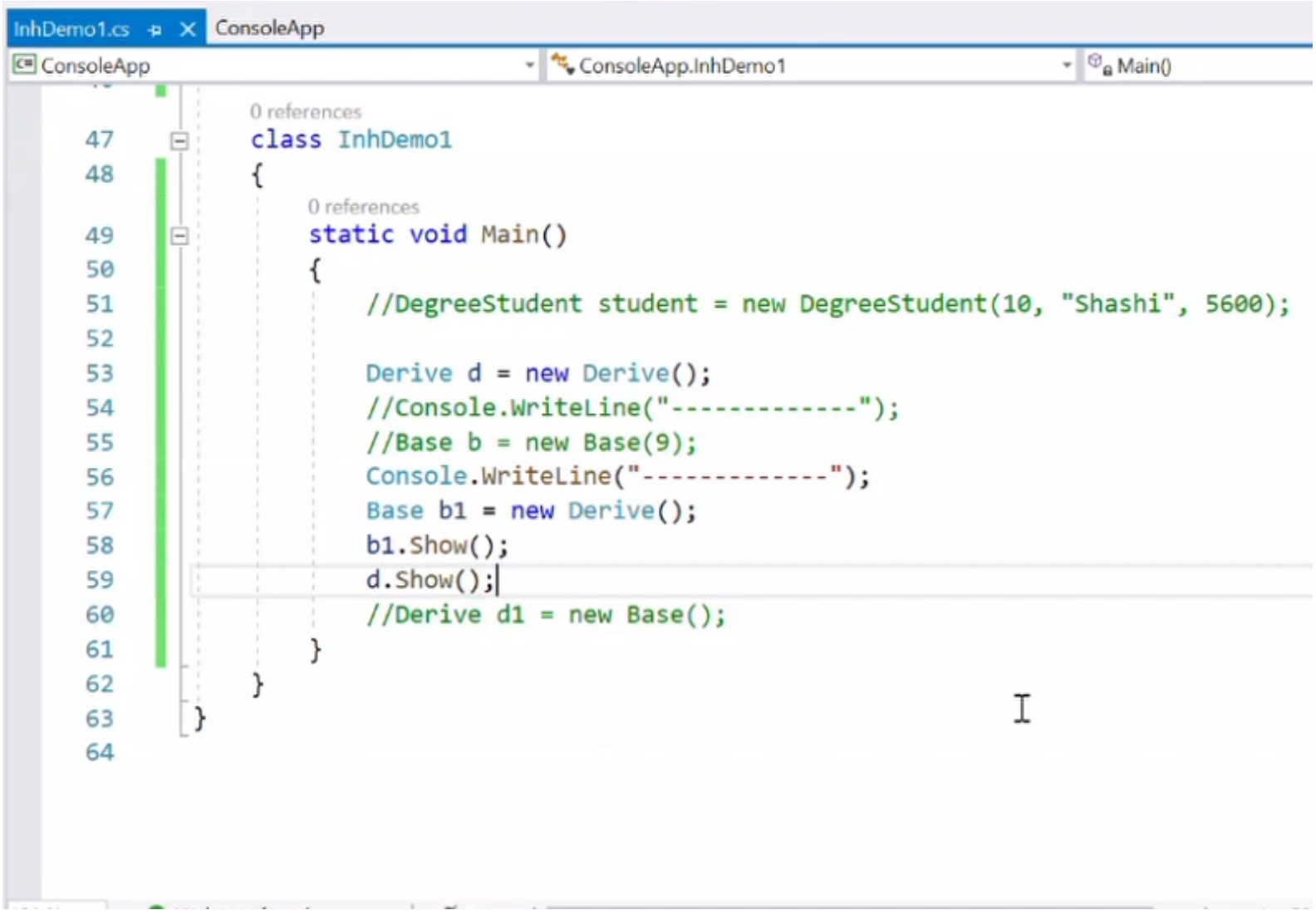


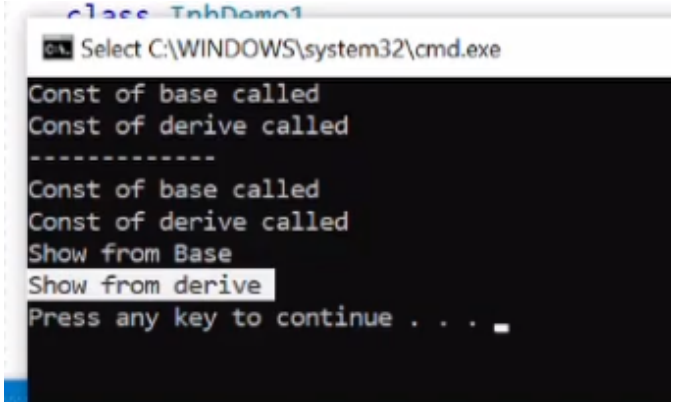
now what will be the output


InhDemo1.cs
ConsoleApp
ConsoleApp
ConsoleApp.InhDemo1
Main()
0 references
class InhDemo1
{
0 references
static void Main()
{
//DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
//Derive d = new Derive();
//Console.WriteLine("-------------");
//Base b = new Base(9);
Console.WriteLine("-------------");
Base b1 = new Derive();
b1.Show();
//Derive d1 = new Base();
}
}
}



now what happens


InhDemo1.cs
ConsoleApp
ConsoleApp
ConsoleApp.InhDemo1
Main()
0 references
class InhDemo1
{
0 references
static void Main()
{
//DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
Derive d = new Derive();
//Console.WriteLine("-------------");
//Base b = new Base(9);
Console.WriteLine("-------------");
Base b1 = new Derive();
b1.Show();
d.Show();
//Derive d1 = new Base();
}
}
}

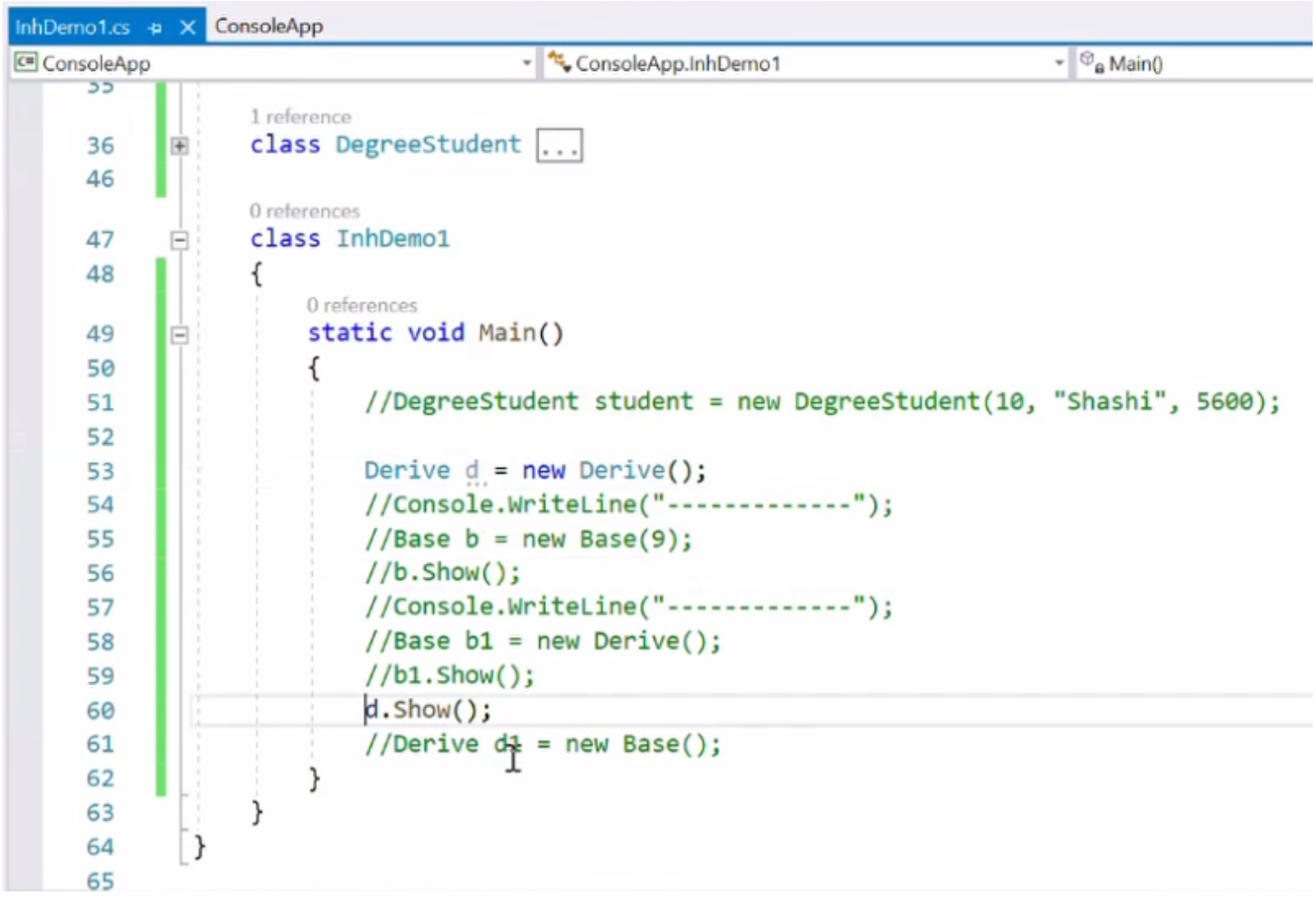


the show method is on for base

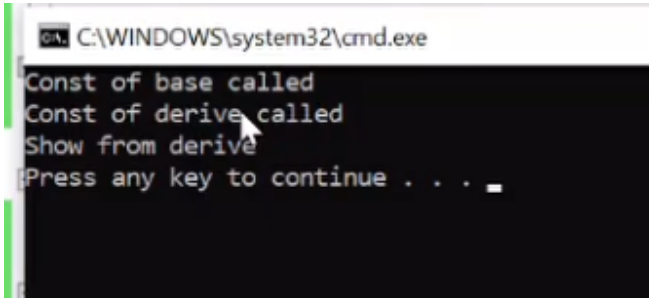
the new show is for derive

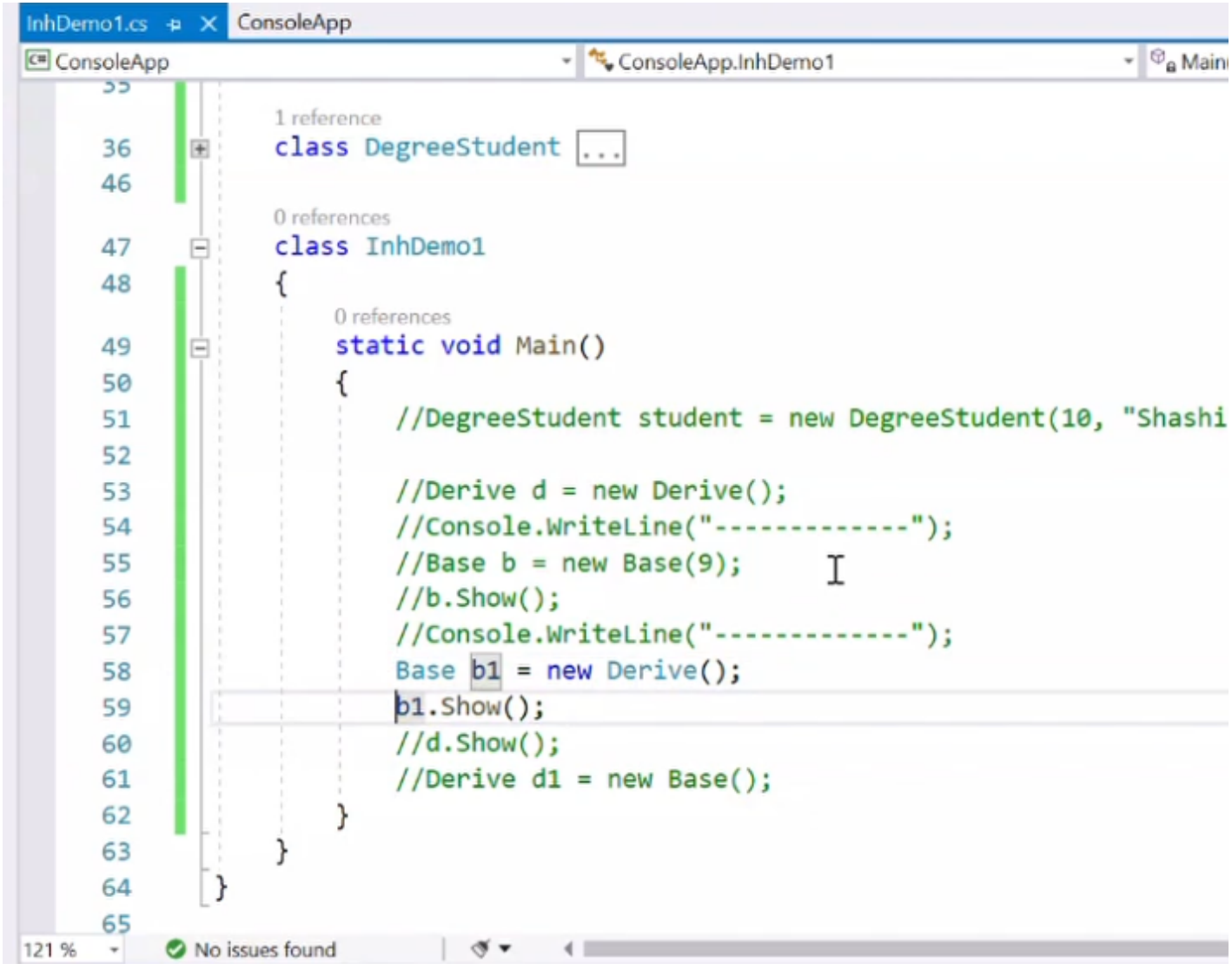
when i create derive instances, i expect the show method of derive get called not the base

In the same way when i create the reference of base class i expect the show of base get called not the derive


InhDemo1.cs
ConsoleApp
ConsoleApp
ConsoleApp.InhDemo1
Main()
1 reference
class DegreeStudent ...
0 references
class InhDemo1
{
0 references
static void Main()
{
//DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
Derive d = new Derive();
//Console.WriteLine("-------------");
//Base b = new Base(9);
//b.Show();
//Console.WriteLine("-------------");
//Base b1 = new Derive();
//b1.Show();
d.Show();
//Derive d1 = new Base();
}
}
}

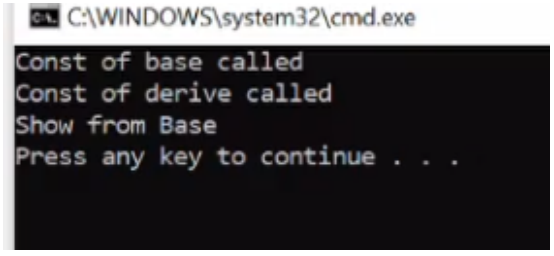
now ?


C:\WINDOWS\system32\cmd.exe
Const of base called
Const of derive called
Show from derive
Press any key to continue . . .


InhDemo1.cs
ConsoleApp
ConsoleApp
ConsoleApp.InhDemo1
1 reference
class DegreeStudent ...
0 references
class InhDemo1
{
0 references
static void Main()
{
//DegreeStudent student = new DegreeStudent(10, "Shashi
//Derive d = new Derive();
//Console.WriteLine("-------------");
//Base b = new Base(9);
//b.Show();
//Console.WriteLine("-------------");
Base b1 = new Derive();
b1.Show();
//d.Show();
//Derive d1 = new Base();
}
}
}
121 %
No issues found

Now?

Here Object Reference Base ka hai, but we are creating object of derive
here it is like dad = new son
and I'm saying like dad has a building
to kiska bulding aega dad ka ki bete ka ?? - ----- dad ka



now whenever somebody ask my property, I want to count my dad's property too there
so for this I'll say **base.show()**

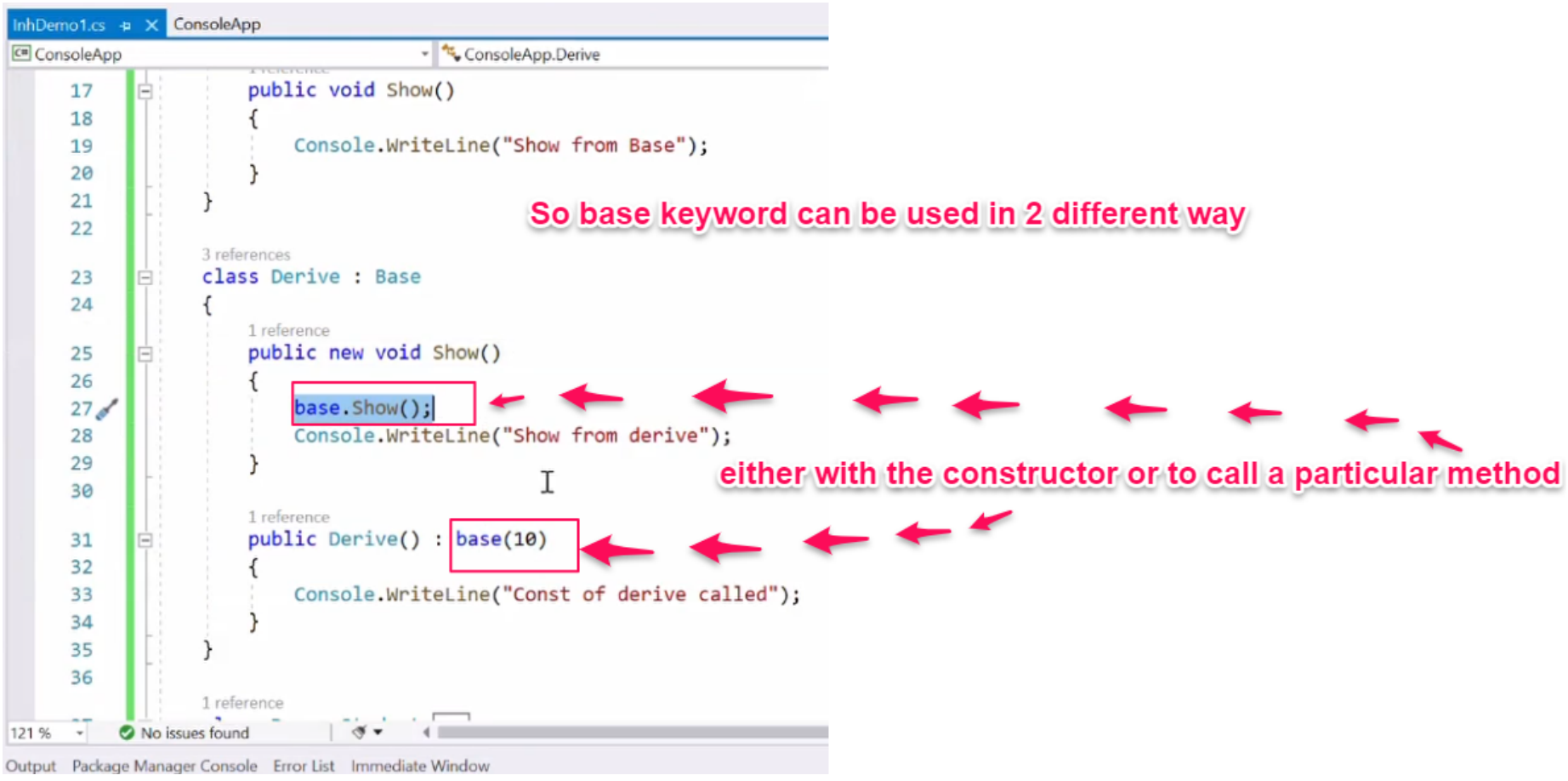
InhDemo1.cs | ConsoleApp

ConsoleApp | ConsoleApp.Derive

```
17          public void Show()
18          {
19              Console.WriteLine("Show from Base");
20          }
21      }
22
        3 references
23      class Derive : Base
24      {
            1 reference
25          public new void Show()
26          {
27              base.Show();
28              Console.WriteLine("Show from derive");
29          }
30
            1 reference
31          public Derive() : base(10)
32          {
33              Console.WriteLine("Const of derive called");
34          }
35      }
36
```

C:\WINDOWS\system32\cmd.exe

```
Const of base called
Const of derive called
Show from Base
Show from derive
Press any key to continue . . .
```


InhDemo1.cs
ConsoleApp
ConsoleApp
ConsoleApp.Derive
public void Show()
{
Console.WriteLine("Show from Base");
}
}
3 references
class Derive : Base
{
1 reference
public new void Show()
{
base.Show();
Console.WriteLine("Show from derive");
}
1 reference
public Derive() : base(10)
{
Console.WriteLine("Const of derive called");
}
}
1 reference
121 %
No issues found
Output
Package Manager Console
Error List
Immediate Window
So base keyword can be used in 2 different way
either with the constructor or to call a particular method

Example: Using 'base' Keyword

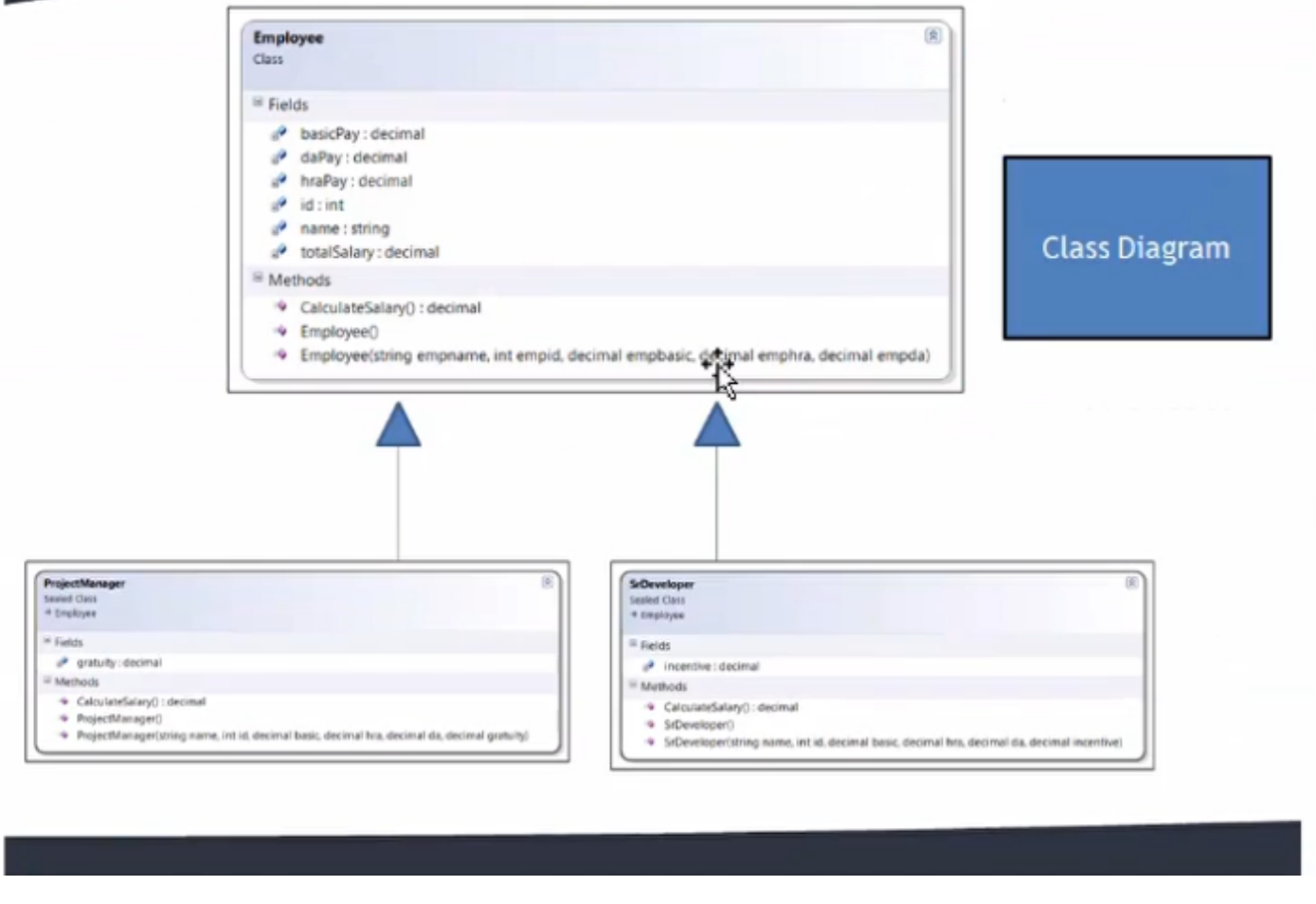
- Scenario:
  - Let's take the same example of InfoSystem Ltd. Company, where we need to calculate salary of different employees

- Problem:
  - Consider, all the common fields in the base class Employee are private. So, no field will be derived in the child classes, SrDeveloper and ProjectManager
  - Also, salary structure of different employees is different.Consider, ProjectManager has 'gartuity' and SrDeveleoper has 'incentive' as their extra payment apart from common 'basic', 'da' and 'hra'. Now you need to calculate salary for both of them but the salary calculation will be slightly different for each derived class.

Example: using 'base' keyword

- Solution:
  - First,
    - Let's add overloaded constructors in base class (Employee) as well as in all derived classes (SrDeveloper and ProjectManager).
    - Call overloaded constructor of derived classes and pass all the required arguments to it, while creating objects of derived classes
    - Form derived class constructor, using 'base' keyword call base class overloaded constructor and pass those same arguments which were passed to derived class overloaded constructor by user
    - Get all the fields be initialized and also salary be calculated in the base class

Example: Using 'base' Keyword



Class Diagram

Example: Using 'base' Keyword

Employee class code

```
class Employee
{
    private string name;
    private int id;
    private decimal basicPay;
    private decimal hraPay;
    private decimal daPay;
    private 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 decimal CalculateSalary()
    {
        this.totalSalary = this.basicPay + this.daPay + this.hraPay;
        return this.totalSalary;
    }
}
```

Example: Using 'new' Keyword

- Scenario:
  - Let's take the same example of InfoSystem Ltd. Company, where we need to calculate salary of different employees
- Problem:
  - Since, salary structure of different employees is different the salary calculation will be slightly different for each derived class. That is why even derived classes had CalculateSalary method with different implementation in the previous solution
  - Since child class is having two CalculateSalary method, where one is derived and another is it's own, hence CalculateSalary method of derived class will try to hide same method of the base class. But, that will not be possible and compiler will be generating an error
- Solution:
  - Use 'new' keyword with the child class CalculateSalary method, so that now child class method can hide the base class method properly and there will be no compiler generated warning

What is Method Overriding?

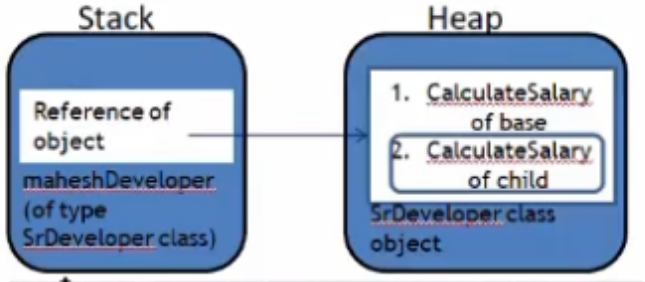
- It is an example of Dynamic Polymorphism
- It is required when you need to provide a new implementation of the base class method in the derived class without changing the signature of the method and also to invoke that child class implementation always, no matter where the reference is passed, either to base or child class variable
- Technique:
  - Declare base class method with 'virtual' keyword
  - Override base class method in derived class by using 'override' keyword with derived class method
  - In this way, in the derived class you are providing new implementation to the same method of base class

Why Method Overriding?

```
static void Main(string[] args)
{
    //information about Senior Developer
    Console.Write("Enter Name of Sr. Developer: ");
    string devname = Console.ReadLine();
    Console.Write("Enter Employee Id: ");
    int devid = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter Basic Payment Amount: ");
    decimal devbasic = Convert.ToDecimal(Console.ReadLine());
    Console.Write("Enter Hra Payment Amount: ");
    decimal devhra = Convert.ToDecimal(Console.ReadLine());
    Console.Write("Enter Da Payment Amount: ");
    decimal devda = Convert.ToDecimal(Console.ReadLine());
    Console.Write("Enter Incentive Payment Amount: ");
    decimal devincentive = Convert.ToDecimal(Console.ReadLine());

    //creating an object of Sr.Developer class and
    //passing the refernce to SrDeveloper class variable
    SrDeveloper maheshDeveloper = new SrDeveloper(
       devname, devid, devbasic, devhra, devda, devincentive);

    //calculating total salary of Sr Developer, Mahesh
    Console.WriteLine("Salary: " + maheshDeveloper.CalculateSalary());
}
```




```
Enter Name of Sr. Developer: Mahesh
Enter Employee Id: 100
Enter Basic Payment Amount: 12000
Enter Hra Payment Amount: 13000
Enter Da Payment Amount: 15000
Enter Incentive Payment Amount: 5000
Salary: 45000
Press any key to continue . . .
```

Notice the salary is 45,000, which is expected output, since derived class method is being called

Why Method Overriding?

- Solution:
  - That means, not only CalculateSalary() has to be implemented in the derived classes with different logic, also CalculateSalary method of derived class should be invoked always, no matter to whom the reference of derived class is passed
- To solve the problem in this scenario, we need help of Method Overriding technique

Example: Method Overriding

Employee class code

```
class Employee
{
   private string name;
   private int id;
   private decimal basicPay;
   private decimal hraPay;
   private decimal daPay;
   private 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 virtual decimal CalculateSalary()
   {
      this.totalSalary = this.basicPay + this.daPay + this.hraPay;
      return this.totalSalary;
   }
}
```

Example: Method Overriding

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()
   {
      decimal basesalary = base.CalculateSalary();
      return (this.gratuity + basesalary);
   }
}
```

```
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()
   {
      decimal basesalary = base.CalculateSalary();
      return (this.incentive + basesalary);
   }
}
```

What is Abstract class and method?

- In the previous scenario, if you do not want to provide implementation logic (code) for CalculateSalary method in base Employee class, then you have to declare the method as abstract and that method should not contain any code
- As a result the Employee class also has to be declared as abstract, since it contains an abstract method

- So, an abstract class is the one, which has at least one abstract member
- And, an abstract member is the one which does not have any code or implementation logic
- Why?
  - Some classes exist solely to be derived from
  - It makes no sense to create instances of these classes
  - These classes are *abstract classes*

Declaring Abstract Class, Method and Property

- Use the abstract keyword while declaring **abstract method**. Also, do not provide any implementation for that method
- Use the abstract keyword while implementing **abstract property**. Also do not implement set and get accessors in a property
- The class has to be declared with **abstract** keyword, too

```
abstract class Employee
{
    public abstract void CalculateSalary();
    public abstract decimal TotalSalary
    {
        set;
        get;
    }
    //other abstract or
    //non-abstract members
}
class Test
{
    static void Main( )
    {
        Employee rajeshEmployee
        =new Employee();
    }
}
```

An abstract class cannot be instantiated

Example: Abstract Class

Employee class code

```
abstract class Employee
{
    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: Abstract Class

Derived class code

By default, abstract members are virtual. So, while implementing them in the derived class, one should use 'override' keyword

```
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 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;
   }
}
```