

C# - 10

The screenshot shows the code editor in Visual Studio with the file `InhDemo1.cs` open. The code defines two classes: `Base` and `Derive`. The `Base` class has a constructor that takes an integer `x` and prints a message to the console. The `Derive` class inherits from `Base` and has its own constructor that also prints a message. The code editor highlights the constructor call in the `Derive` constructor with a yellow bar.

```
10 6 references
11 class Base
12 {
13     2 references
14     public Base(int x)
15     {
16         Console.WriteLine("Const of base called");
17     }
18
19 4 references
20 class Derive : Base
21 {
22     2 references
23     public Derive() : base(10)
24     {
25         Console.WriteLine("Const of derive called");
26     }
27 }
```

By default it will always have this signature

```
InhDemo1.cs*  X  ConsoleApp
ConsoleApp  ConsoleApp.Base  Base()

10      6 references
11      class Base
12      {
13          2 references
14          public Base()
15          {
16              Console.WriteLine("Const of base called");
17          }
18      4 references
19      class Derive : Base
20      {
21          2 references
22          public Derive() : base()
23          {
24              Console.WriteLine("Const of derive called");
25          }
26      }
```

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

```
InhDemo1.cs*  X  ConsoleApp
ConsoleApp
ConsoleApp.Base
6 references
class Base
{
    2 references
    public Base(int x)
    {
        Console.WriteLine("Const of base called");
    }
}

4 references
class Derive : Base
{
    2 references
    public Derive() : base() 
    {
        Console.WriteLine("Const of derive called");
    }
}
```

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

The screenshot shows a code editor window for a C# project named 'ConsoleApp'. The file 'InhDemo1.cs' is open. The code defines three classes: 'Base', 'Derive', and 'DegreeStudent'. The 'DegreeStudent' class inherits from 'Student'. A tooltip for 'DegreeStudent' indicates it has 6 references. Another tooltip for 'Student' indicates it has 4 references. The code completion interface is visible, showing the inheritance relationship between 'DegreeStudent' and 'Student'.

```
10 6 references  
11 class Base ...  
12  
13 4 references  
14 class Derive ...  
15  
16 class DegreeStudent : Student  
17 {  
18 }  
19  
20 }
```

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

InhDemo1.cs* ConsoleApp

ConsoleApp ConsoleApp.Student

```
8     {
9     ^ 12 references
10    class Student
11    {
12        5 references
13        public int id { get; set; }
14        5 references
15        public string name { get; set; }
16        3 references
17        public Student(int id)
18        {
19            if(id > 5)
20            {
21                id = id + 5;
22            }
23            this.id = id;
24            Console.WriteLine("0 para const");
25        }
26
27        1 reference
28        public Student(int id, string name):this(id)
29        {
30            this.name = name;
31            Console.WriteLine("1 para const");
32        }
33    }
34 }
```

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

```
InhDemo1.cs*  ✘ X  ConsoleApp
ConsoleApp  ✘ ConsoleApp.DegreeStudent
10 6 references
class Base...
17
18 4 references
class Derive ...
25
26 1 reference
class DegreeStudent : Student
27 {
28 0 references
public DegreeStudent() : base() I
29 {
30 }
31 }
32 }
33 0 references
```

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

InhDemo1.cs* X ConsoleApp

ConsoleApp

ConsoleApp.DegreeStudent

DegreeStudent(int id, string name)

```
10 class Base{...}
17
18 class Derive{...}
25
26 class DegreeStudent : Student
27 {
28     public DegreeStudent(int id, string name) : base()
29     {
30     }
31 }
32
33
34 class InhDemo1
35 {
36     static void Main()
37     {
38         Derive d = new Derive();
39         Console.WriteLine("-----");
40         Base b = new Base();
```

see I have 2 constructor here

The screenshot shows a Microsoft Visual Studio code editor with the file 'InhDemo1.cs' open. The cursor is at the end of the line 'public DegreeStudent(int id, string name) : base()'. A tooltip from the code completion feature is displayed, showing '▲ 1 of 2 ▼ Student(int id)' with a red arrow pointing to it. Below the tooltip is a dropdown menu with 'base' selected. The code editor interface includes a status bar at the bottom showing 'Ln: 28 Ch: 52 Col: 58'.

InhDemo1.cs* X ConsoleApp

PassbyValueDemo2.cs

ConsoleApp

ConsoleApp.DegreeStudent

DegreeStudent(int id, string name)

```
10 6 references
11 class Base{...}
12
13 4 references
14 class Derive{...}
15
16 1 reference
17 class DegreeStudent : Student
18 {
19     0 references
20     public DegreeStudent(int id, string name) : base()
21     {
22     }
23 }
24
25 0 references
26 class InhDemo1
27 {
28     0 references
29     static void Main()
30     {
31         Derive d = new Derive();
32         Console.WriteLine("-----");
33         Base b = new Base();
```

▲ 2 of 2 ▼ Student(int id, string name)

- await
- BadImageFormatException
- Base
- base
- Base64FormattingOptions
- BitConverter
- bool
- Boolean

The screenshot shows a Microsoft Visual Studio code editor window. The main pane displays C# code for a class named 'DegreeStudent' which inherits from 'Student'. A tooltip is open over the 'base()' constructor call, showing two options: 'base()' and 'Student(int id, string name)'. The 'Student(int id, string name)' option is highlighted. To the right of the tooltip is a standard Windows-style context menu with icons for copy, cut, paste, etc.

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**

```
InhDemo1.cs*      ConsoleApp
ConsoleApp          ConsoleApp.Student
10     {
11         public int id { get; set; }
12         public string name { get; set; }
13         public Student(int id)
14         {
15             if(id > 5)
16             {
17                 id = id + 5;
18             }
19             this.id = id;
20             Console.WriteLine("0 para const");
21         }
22
23         public Student(int id, string name):this(id)
24         {
25             this.name = name;
26             Console.WriteLine("1 para const");
27         }
28
29 }
```

So as I'm passing 2 parameter
so this program will called and in return this will be called
and the id gets initialized and name gets initialized here

then it will come back to this program

InhDemo1.cs* X ConsoleApp

ConsoleApp

ConsoleApp.DegreeStudent

Degree

```
10 6 references
class Base{...}

17
18 4 references
class Derive {...}

25
26 1 reference
class DegreeStudent : Student
{
27
28 0 references
    public DegreeStudent(int id, string name) : base(id, name)
29    {
30        I
31    }
32}
33 0 references
```

Suppose I have a double fee and

InhDemo1.cs* X ConsoleApp PassbyValueDer

ConsoleApp

ConsoleApp.InhDemo1

Main()

```
26 class DegreeStudent : Student
27 {
28     double fees;
29     public DegreeStudent(int id, string name, double fees)
30         : base(id, name)
31     {
32         this.fees = fees;
33         Console.WriteLine("Const of Degree student");
34     }
35 }
36
37 class InhDemo1
38 {
39     static void Main()
40     {
41         DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
42
43         Derive d = new Derive();
44         Console.WriteLine("-----");
45         Base b = new Base(9);
46         Console.WriteLine("-----");
47     }
48 }
```

0 references

0 references

1 reference

121 % No issues found

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

InhDemo1.cs ConsoleApp

ConsoleApp ConsoleApp.Student

```
10  {
11      5 references
12      public int id { get; set; }
13      5 references
14      public string name { get; set; }
15      3 references
16      public Student(int id)
17      {
18          if(id > 5)
19          {
20              id = id + 5;
21          }
22      }
23      2 references
24      public Student(int id, string name):this(id)
25      {
26          this.name = name;
27          Console.WriteLine("1 para const");
28      }
29
30 }
```

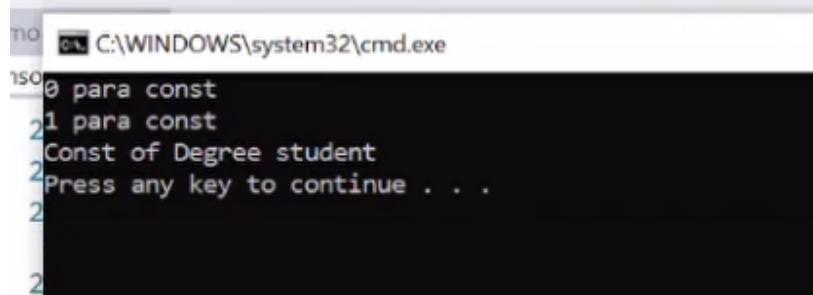
121 % No issues found

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**



```
C:\WINDOWS\system32\cmd.exe
1sc
0 para const
1 para const
Const of Degree student
Press any key to continue . . .
2
2
```

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

```
class Date
{
    public Date( ) <----- Default constructor
    {
        ...
    }

    public Date(int year, int month, int day)
    {
        ...
    }
}
```

The diagram illustrates the code for the `Date` class. It shows two constructor definitions. The first constructor, which takes no parameters, is labeled "Default constructor". The second constructor, which takes three parameters (`int year, int month, int day`), is labeled "Overloaded constructor". Arrows point from the labels to their respective constructor definitions in the code.

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;
    }
}
```

Static Constructor

Explicit Default Constructor

Overloaded Constructor

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

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `InhDemo1.cs` under the tab `ConsoleApp`. The code defines two classes: `Base` and `Derive`. The `Base` class has a constructor `public Base(int x)` and a method `Show()`. The `Derive` class inherits from `Base` and overrides the `Show()` method. A green squiggly underline is underlined under the `Show()` method in the `Derive` class, indicating a warning or error. The Solution Explorer window on the right lists various demo files for the `ConsoleApp` project, including `Calc.cs`, which is currently selected.

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

```
12     public Base(int x)
13     {
14         Console.WriteLine("Const of base called");
15     }
16
17     public void Show()
18     {
19         Console.WriteLine("Show from Base");
20     }
21
22
23     class Derive : Base
24     {
25         public new void Show()
26         {
27             Console.WriteLine("Show from derive");
28         }
29
30         public Derive() : base(10)
31     }

```

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

A screenshot of the Microsoft Visual Studio IDE interface. The main window shows the code editor with two tabs: "InhDemo1.cs*" and "ConsoleApp". The "InhDemo1.cs*" tab contains C# code demonstrating inheritance:

```
7 namespace ConsoleApp
8 {
9
10    class Base
11    {
12        public Base(int x)
13        {
14            Console.WriteLine("Const of base called");
15        }
16
17        public void Show()
18        {
19            Console.WriteLine("Show from Base");
20        }
21    }
22
23    class Derive : Base
24    {
25        public new void Show()
26        {
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121 }
```

The Solution Explorer on the right lists files in the "ConsoleApp" project, including "Calc.cs" which is currently selected.

At the bottom, the status bar displays: "121 %", "No issues found", "Ln: 59 Ch: 13 TABS CRLF".

```
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



```
InhDemo1.cs*  ConsoleApp  PassbyValueDe
ConsoleApp  ConsoleApp.InhDemo1  Main()
1 reference
36 class DegreeStudent ...
0 references
37 class InhDemo1
38 {
39     0 references
40     static void Main()
41     {
42         //DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
43
44         //Derive d = new Derive();
45         //Console.WriteLine("-----");
46         Base b = new Base(9);
47         Console.WriteLine("-----");
48         //Base b1 = new Derive();
49         b.Show();
50
51         //Derive d1 = new Base();
52     }
53 }
```

what will be the output of this

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

now what will be the output

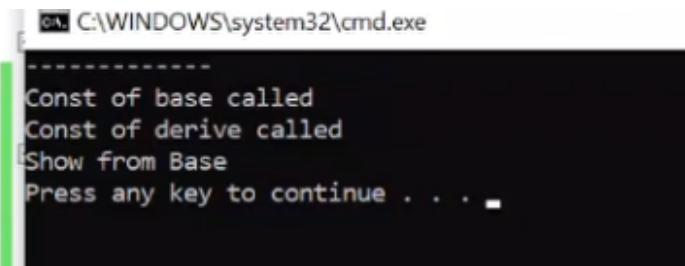
InhDemo1.cs X ConsoleApp

ConsoleApp

ConsoleApp.InhDemo1

Main()

```
47 0 references
48 class InhDemo1
49 {0 references
50     static void Main()
51     {
52         //DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
53
54         //Derive d = new Derive();
55         //Console.WriteLine("-----");
56         //Base b = new Base(9);
57         Console.WriteLine("-----");
58         Base b1 = new Derive();
59         b1.Show();
60
61         //Derive d1 = new Base();
62     }
63 }
64 }
```



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

now what happens

InhDemo1.cs X ConsoleApp

ConsoleApp ConsoleApp.InhDemo1 Main()

```
47  class InhDemo1
48  {
49      static void Main()
50      {
51          //DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
52
53          Derive d = new Derive();
54          //Console.WriteLine("-----");
55          //Base b = new Base(9);
56          Console.WriteLine("-----");
57          Base b1 = new Derive();
58          b1.Show();
59          d.Show();
60          //Derive d1 = new Base();
61      }
62  }
63
64
```

```
class TestDemo1  
{  
public:  
    TestDemo1() { cout << "Const of base called" << endl; }  
    void show() { cout << "Show from Base" << endl; }  
protected:  
    TestDemo1() { cout << "Const of derive called" << endl; }  
    void show() { cout << "Show from derive" << endl; }  
private:  
};  
  
int main()  
{  
    TestDemo1 t1;  
    t1.show();  
    return 0;  
}
```

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 X ConsoleApp

ConsoleApp ConsoleApp.InhDemo1 Main()

```
36 class DegreeStudent { ... }
46
47 class InhDemo1
48 {
49     static void Main()
50     {
51         //DegreeStudent student = new DegreeStudent(10, "Shashi", 5600);
52
53         Derive d = new Derive();
54         //Console.WriteLine("-----");
55         //Base b = new Base(9);
56         //b.Show();
57         //Console.WriteLine("-----");
58         //Base b1 = new Derive();
59         //b1.Show();
60         d.Show();
61         //Derive d1 = new Base();
62     }
63 }
64
65 }
```

now ?

0. C:\WINDOWS\system32\cmd.exe

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

InhDemo1.cs X ConsoleApp

ConsoleApp

ConsoleApp.InhDemo1

Main

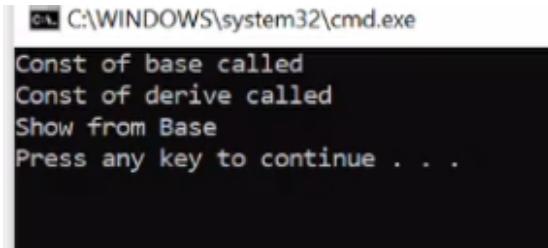
```
36     1 reference
37     class DegreeStudent { ... }
38
39     0 references
40     class InhDemo1
41     {
42         0 references
43         static void Main()
44         {
45             //DegreeStudent student = new DegreeStudent(10, "Shashi
46
47             //Derive d = new Derive();
48             //Console.WriteLine("-----");
49             //Base b = new Base(9);      I
50             //b.Show();
51             //Console.WriteLine("-----");
52             Base b1 = new Derive();
53             b1.Show();
54             //d.Show();
55             //Derive d1 = new Base();
56
57         }
58     }
59 }
```

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



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

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 ✘ X ConsoleApp

ConsoleApp

ConsoleApp.Derive

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

3 references

1 reference

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 + X ConsoleApp

ConsoleApp

```
public void Show()
{
    Console.WriteLine("Show from Base");
}

3 references
class Derive : Base
{
    public new void Show()
    {
        base.Show(); // Call to base class Show() method
        Console.WriteLine("Show from derive");
    }

    public Derive() : base(10) // Constructor call to base class
    {
        Console.WriteLine("Const of derive called");
    }
}
```

So base keyword can either

121% ✓ No issues found

So base keyword can be used in 2 different way

either with the constructor or to call a particular method

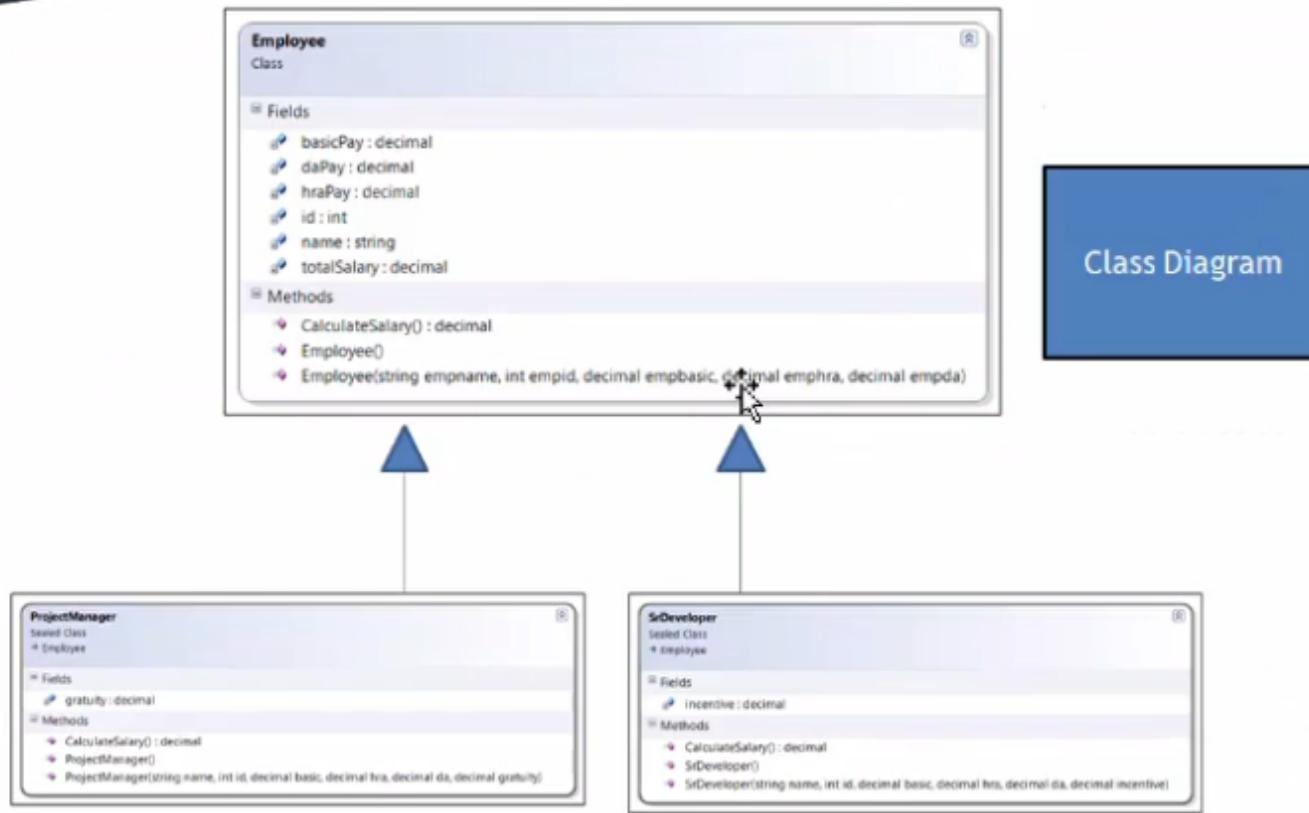
Example: Using 'base' Keyword

- 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



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?

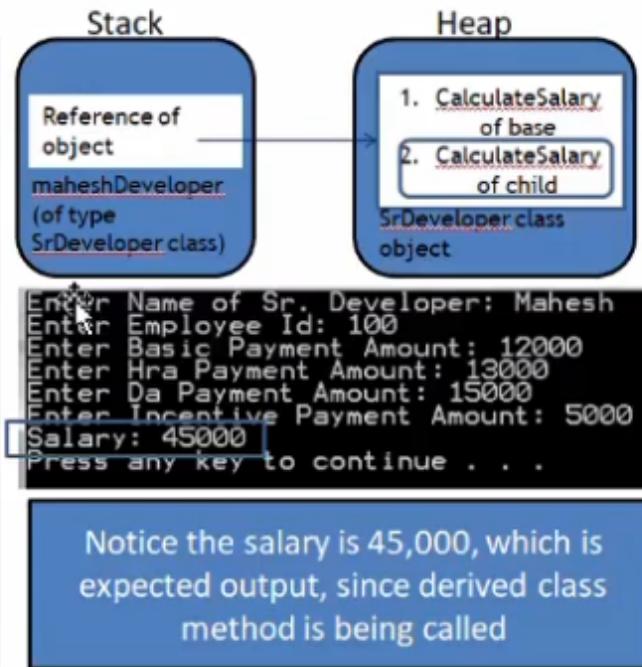
- It is an example of Dynamic Polymorphism
- It is required when you need to provide a new implementation of the base class method in the derived class without changing the signature of the method and also to invoke that child class implementation always, no matter where the reference is passed, either 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.WriteLine("Enter Name of Sr. Developer:");
    string devname = Console.ReadLine();
    Console.WriteLine("Enter Employee Id:");
    int devid = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter Basic Payment Amount:");
    decimal devbasic = Convert.ToDecimal(Console.ReadLine());
    Console.WriteLine("Enter Hra Payment Amount:");
    decimal devhra = Convert.ToDecimal(Console.ReadLine());
    Console.WriteLine("Enter Da Payment Amount:");
    decimal devda = Convert.ToDecimal(Console.ReadLine());
    Console.WriteLine("Enter Incentive Payment Amount:");
    decimal devincentive = Convert.ToDecimal(Console.ReadLine());

    //creating an object of Sr.Developer class and
    //passing the reference 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());
}
```



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(); ← X
    }
}
```

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;
    }
}
```