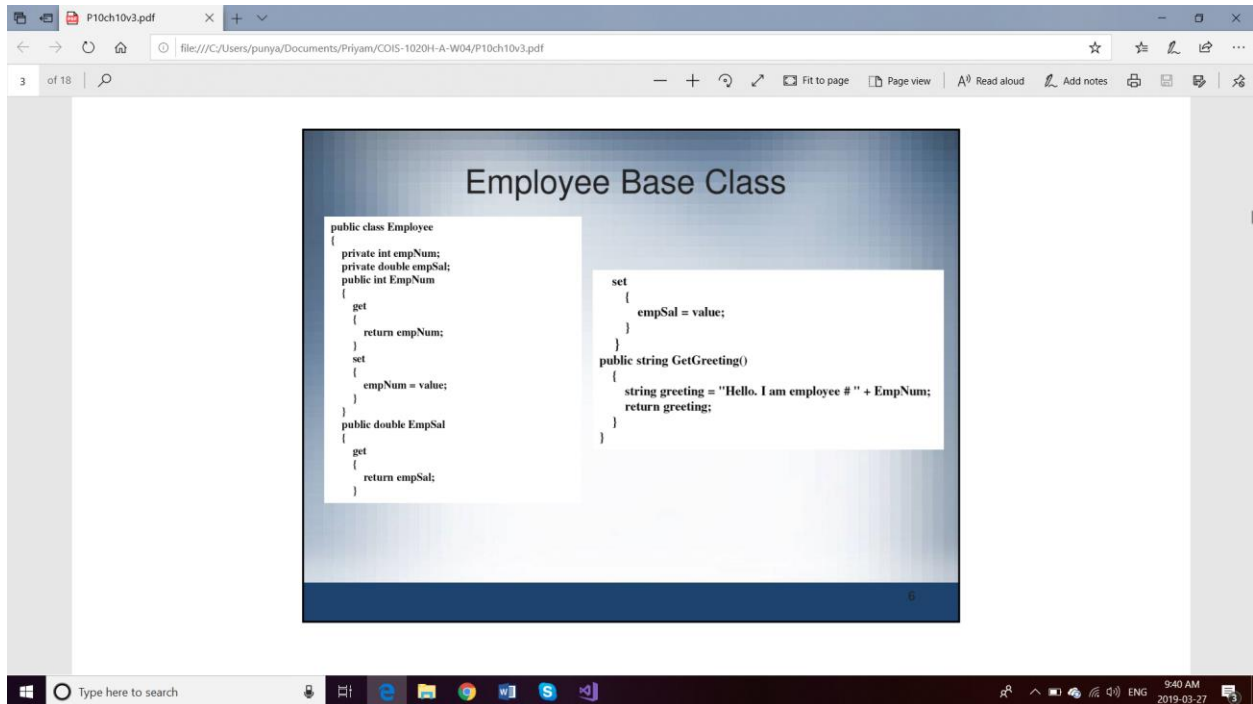
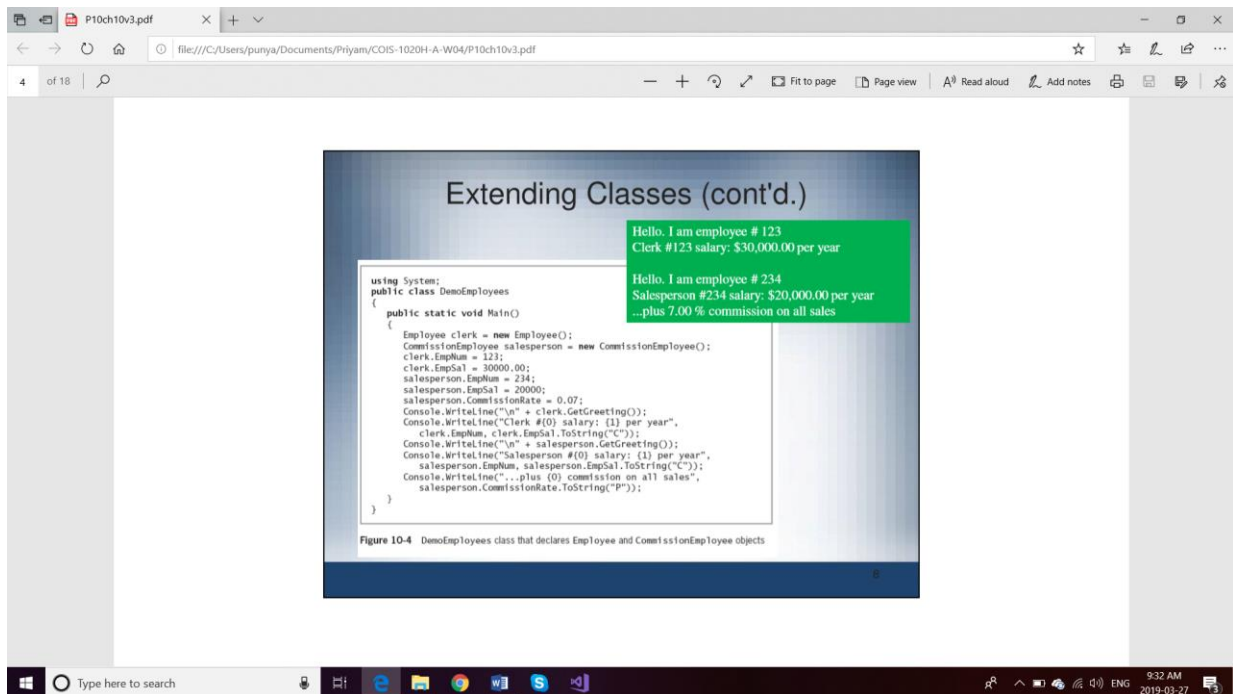
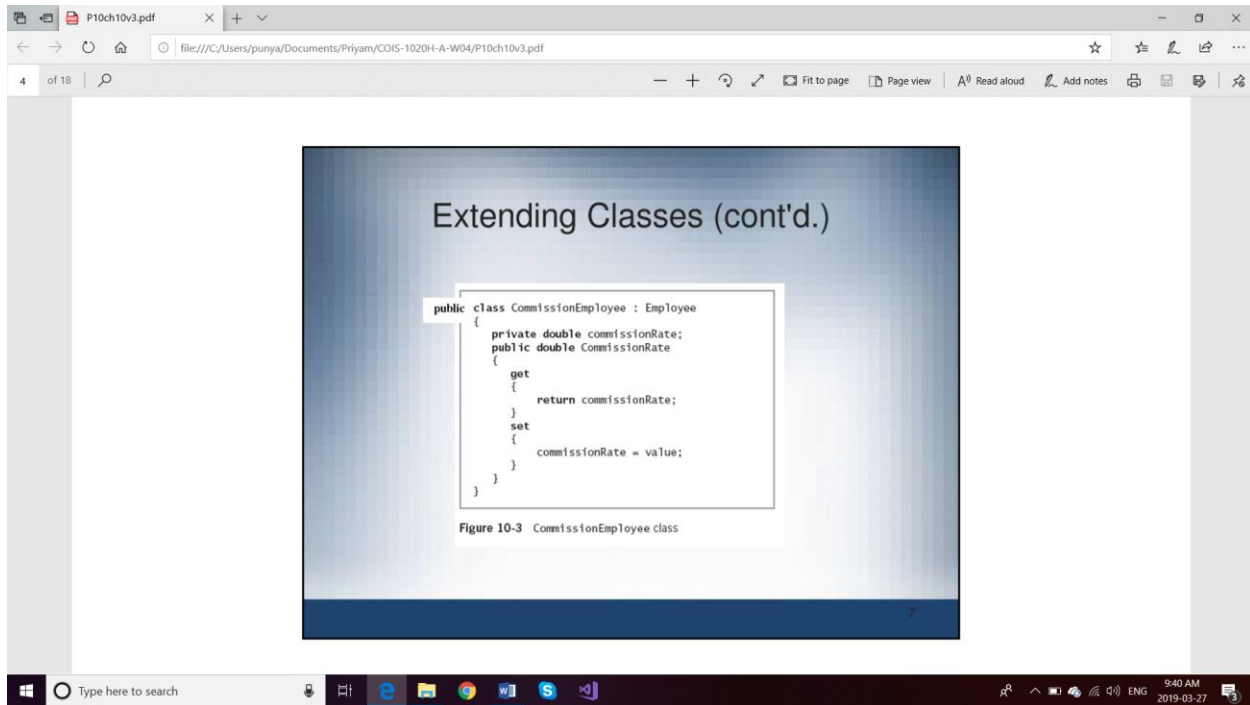


INHERITANCE

- (1) The DERIVED CLASS is the inherited class. And sad as “a truck ‘is a’ vehicle”
- (2) Syntax –
Truck : vehicle
- (3) A DERIVED CLASS gets direct access to anything public in the BASE CLASS
- (4) If something is declared in the Derived class, then that parameters plus the public ones in the base class can be seen by the derived class.



All the public parameters In the Employee class will be accessible by the CommissionEmployee class



In the beginning of the program, clerk stores null. Then it stores Employee.

Similarly, the salesperson, stores CommissionEmployee.

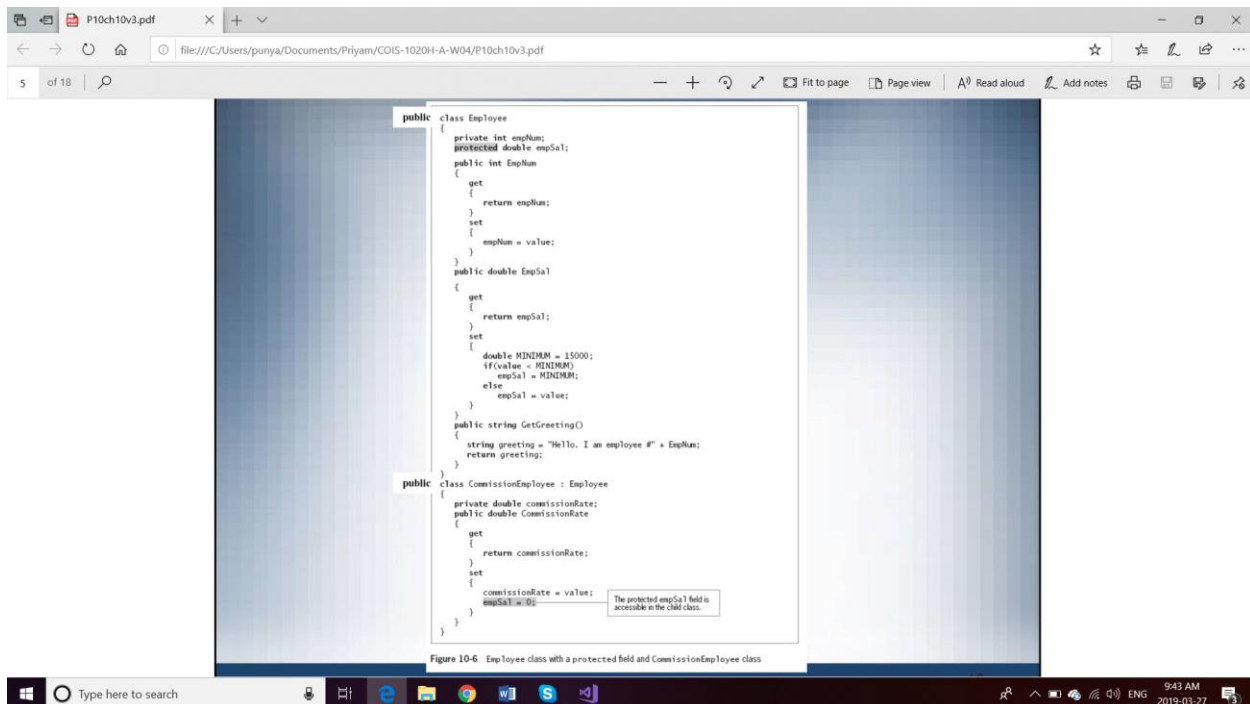
From point of view of Main(), I can assign three different things to the salesperson, because I has access to both Employee class and CommissionEmployee class. And no specific order needs to be followed while assigning values.

That is why, we've assigned values to salesperson.EmpNum, salesperson.EmpSal and salesperson.CommissionRate

.empNum cannot be done since it is private. It has to be .EmpNum

PROTECTED access specifier

A protected parameter can be used by the class it is in and an extended class only.



If instead of protected, we use private, then we get error, that it is inaccessible due to its protection level.

Now if there is a GetGreeting() Method in the CommissionEmployee class, and when in Main() we call GetGreeting() Method, the compiler goes to the closest one, that means in the derived class and not the parent/base class.

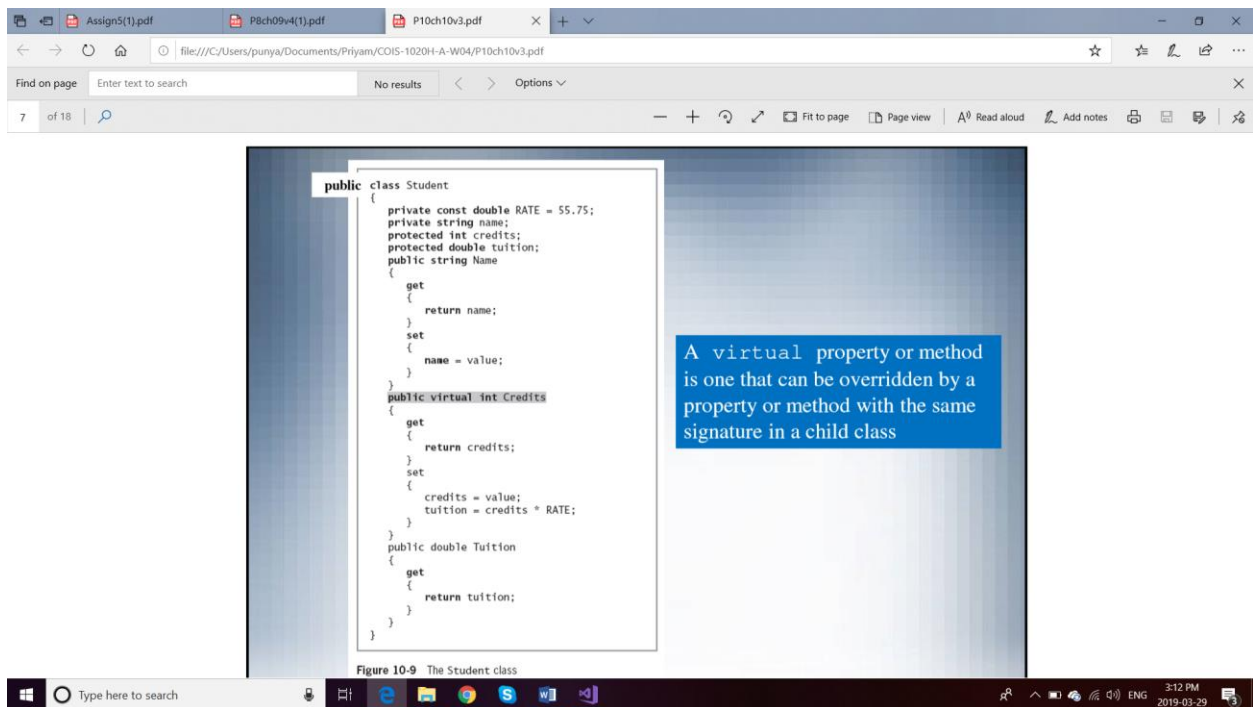
If I do, base.GetGreeting() in the GetGreeting() in the child class, then the control shifts in the base class GetGreeting() method.

But there is a warning message by C# that there are two methods under the same name. So to remove that, just add new operator in front of the GetGreeting()

OVERRIDE

Virtual is added, to say that the object can be overridden. That means, for parent class it means something else, and something else for a child class. Any Method or can be written virtual if it wants to be overridden. **A virtual method CAN be overridden.**

An abstract method HAS to be overridden



This is the base/parent class

Assign5(1).pdf P8ch09v4(1).pdf P10ch10v3.pdf

file:///C:/Users/punya/Documents/Priyam/COIS-1020H-A-W04/P10ch10v3.pdf

Find on page Enter text to search No results Options

7 of 18

Overriding Base Class Methods (cont'd.)

```
public class ScholarshipStudent : Student
{
    override public int Credits
    {
        set
        {
            credits = value;
            tuition = 0;
        }
    }
}
```

The `override` modifier overrides and "hides" its counterpart in the parent class.

Figure 10-10 The ScholarshipStudent class

Type here to search

3:13 PM 2019-03-29

The child class overrides credits.

Assign5(1).pdf P8ch09v4(1).pdf P10ch10v3.pdf

file:///C:/Users/punya/Documents/Priyam/COIS-1020H-A-W04/P10ch10v3.pdf

Find on page Enter text to search No results Options

8 of 18

```
using System;
class DemoStudents
{
    public static void Main()
    {
        Student payingStudent = new Student();
        ScholarshipStudent freeStudent = new ScholarshipStudent();
        payingStudent.Name = "Megan";
        payingStudent.Credits = 15;
        freeStudent.Name = "Luke";
        freeStudent.Credits = 15;
        Console.WriteLine("{0}'s tuition is {1}",
            payingStudent.Name,
            payingStudent.Tuition.ToString("C"));
        Console.WriteLine("{0}'s tuition is {1}",
            freeStudent.Name,
            freeStudent.Tuition.ToString("C"));
    }
}
```

Megan's tuition is \$836.25
Luke's tuition is \$0.00

Figure 10-11 The DemoStudents program

15

Type here to search

3:13 PM 2019-03-29

The Main() Method.

The 'no argument constructors' are being used here.

Accessing Base Class Methods from a Derived Class

- Use the keyword `base` to access the parent class method

```
public class CommissionEmployee : Employee
{
    private double commissionRate;
    public double CommissionRate
    {
        get
        { return commissionRate; }
        set
        { commissionRate = value;
          empSal = 0;
        }
    }
    new public string GetGreeting()
    {
        string greeting = base.GetGreeting();
        greeting += "\nI work on commission.";
        return greeting;
    }
}
```

The `base` keyword allows a child to access its parent's methods

16

The 'new' keyword has to be used because we used a method with the same name in the base class.

Also, `base`. Is used to indicate which method is being called.

Employee Base Class

```
public class Employee
{
    private int empNum;
    private double empSal;
    public int EmpNum
    {
        get
        { return empNum;
        }
        set
        { empNum = value;
        }
    }
    public double EmpSal
    {
        get
        { return empSal;
        }
    }
}
```

```
set
{
    empSal = value;
}
public string GetGreeting()
{
    string greeting = "Hello. I am employee # " + EmpNum;
    return greeting;
}
```

Understanding How a Derived Class Object "is an" Instance of the Base Class (cont'd.)

```

using System;
public class DemoSalesperson3
{
    public static void Main()
    {
        Employee clerk = new Employee();
        CommissionEmployee salesperson = new CommissionEmployee();
        clerk.EmpNum = 234;
        salesperson.EmpNum = 345;
        DisplayGreeting(clerk);
        DisplayGreeting(salesperson);
    }

    public static void DisplayGreeting(Employee emp)
    {
        Console.WriteLine("Hi there from #" + emp.EmpNum);
        Console.WriteLine(emp.GetGreeting());
    }
}

```

The first call passes an Employee object but the second call passes a CommissionEmployee object

Hi there from #234
Hello. I am employee # 234
Hi there from #345
Hello. I am employee # 345

Figure 10-16 The DemoSalesperson3 program

In here, we are able to pass both the objects of Employee class as well as CommissionEmployee class because they are parent child. However nothing can be accessed from the child class in the method because it is treating it as a parent class i.e. Employee class object.

Using the Object Class (cont'd.)

Method	Explanation
Equals()	Determines whether two Object instances are equal
GetHashCode()	Gets a unique code for each object; useful in certain sorting and data management tasks
GetType()	Returns the type, or class, of an object
<u>ToString()</u>	Returns a String that represents the object

Table 10-1 The four public instance methods of the Object class

ToString() method in the Object class is automatically called. It turns something into a string.

It prints out the details of the object.

Console.WriteLine(sStudent) is same as Console.WriteLine(sStudent.ToString())

We can do much cool things with ToString() method.

The slide is titled "Using the Object Class's ToString() Method". It contains a bulleted list and a code snippet. The list includes: "ToString() method" with a sub-point "Returns a string that holds the class name" (noted as "Just as GetType() does"), and "You should override this method in your own class to make it more meaningful (Employee: 1234 Hurley)". The code snippet shows a C# method:

```
public override string ToString()
{
    return(getType() + ": " + EmpNum + " " + Name);
}
```

 Red curly braces are drawn around the code block. The slide is labeled "Figure 10-20 An Employee class ToString() method". The browser window shows multiple tabs and a taskbar at the bottom.

Using the Object Class's ToString() Method

- ToString() method
 - Returns a string that holds the class name
 - Just as GetType() does
- You should override this method in your own class to make it more meaningful (Employee: 1234 Hurley)

```
public override string ToString()
{
    return(getType() + ": " + EmpNum + " " + Name);
}
```

Figure 10-20 An Employee class ToString() method

The slide is titled "Working with Base Class Constructors". It contains a bulleted list and a code snippet. The list includes: "Instantiating an object of a derived class" with sub-points "Calls the constructor for both the base class and the derived class" and "The base class constructor must execute first". The code snippet shows two C# classes:

```
public class Employee
{
    private int empNum;
    protected double empSal;
    public Employee()
    {
        Console.WriteLine("Employee constructed");
    }
}

public class CommissionEmployee : Employee
{
    private double commissionRate;
    public CommissionEmployee()
    {
        Console.WriteLine("CommissionEmployee constructed");
    }
}
```

 The slide is labeled "Figure 10-24 Employee and CommissionEmployee classes with parameterless constructors". The browser window shows a single tab and a taskbar at the bottom.

Working with Base Class Constructors

- Instantiating an object of a derived class
 - Calls the constructor for both the base class and the derived class
 - The base class constructor must execute first

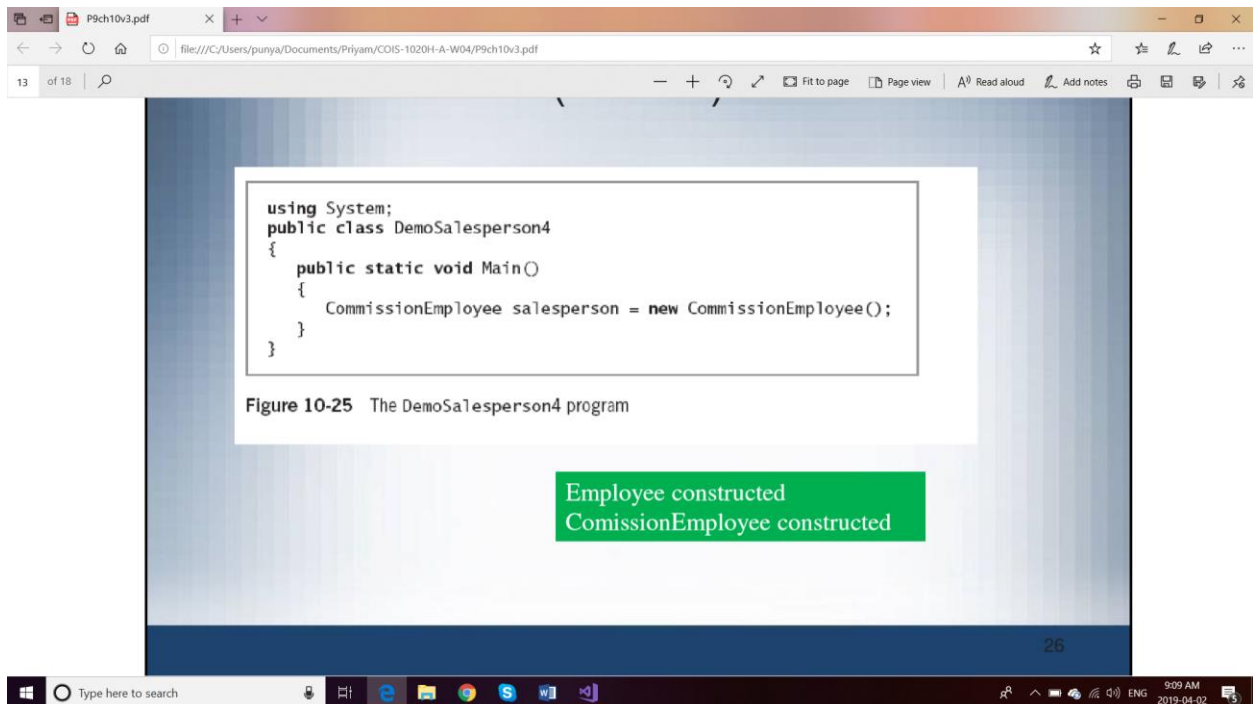
```
public class Employee
{
    private int empNum;
    protected double empSal;
    public Employee()
    {
        Console.WriteLine("Employee constructed");
    }
}

public class CommissionEmployee : Employee
{
    private double commissionRate;
    public CommissionEmployee()
    {
        Console.WriteLine("CommissionEmployee constructed");
    }
}
```

Figure 10-24 Employee and CommissionEmployee classes with parameterless constructors

When a child constructor is constructed, first the base constructor is invoked then the child.

That means :



If a base class requires arguments then every derived constructor when created, needs to pass constructor.

If base constructor is : `public Employee(int empId, string empName)`

Using Base Class Constructors that Require Arguments (cont'd)

- Examples:

```
public CommissionEmployee() : base(1234, "XXXX")
{ Other statements go here }
- CommissionEmployee constructor requires no arguments but the base class
Employee constructor requires 2
```

```
public CommissionEmployee(int id, string name) : base(id, name)
{ Other statements go here }
- CommissionEmployee constructor requires 2 arguments and it passes these
on to the base class Employee constructor
```

```
public CommissionEmployee(int id, string name, double rate) :
base(id, name)
{ CommissionRate = rate; Other statements go here }
- CommissionEmployee constructor requires 3 arguments, passes 2 to the base
class Employee constructor and uses the other for itself
```

14 of 18

In here, the first one, we have made a derived constructor with no argument, thus we pass the arguments using base.

ABSTRACT CLASS

- No object instantiated.
- It consists of the generic information, from which other classes can inherit information from.
- Can contain virtual methods and also abstract methods.
- Abstract Methods : anyone inheriting from this class, must have this method

```
public abstract class Animal
{
    private string name;
    public Animal (string valName)
    {
        name = valName;
    }
    public string Name
    {
        get
        {
            return name;
        }
    }
    public abstract string Speak();
}
```

30

```
public class Dog : Animal
{
    public Dog(string name) : base(name)
    {
    }
    public override string Speak()
    {
        return "woof";
    }
}

public class Cat : Animal
{
    public Cat(string name) : base(name)
    {
    }
    public override string Speak()
    {
        return "meow";
    }
}
```

Figure 10-28 Dog and Cat classes

There is a ; after speak() method which means the derived class has to provide the contents in the speak else the Animal doesn't speak.

In the abstract class, we can write

```
public Animal(string name)
{
    this.name=name;
}
```

//what we are doing here is that, we use the same name, which is confusing and thus we have to use this.name because this reference has to be used to tell which name is the object one to which we are giving the value name.

If we don't override an abstract method, it shows an error that the derived class doesn't inherit the abstract method.

If we make the abstract method into a virtual method, then we obviously have to contain some statements in it. Let's say :

```
public virtual Speak() { return("Hello");}
```

then, if any derived class has overridden this method, then that method runs else, the virtual will run. That is if there is no override speak method in derived class, then the original that is the virtual method is run.

Also, if we try to make an object in an abstract class then error is shown.

Let's say.

Abstract class : Student

- Name
- Tuition
- Credits

Derived class : International student

- Country

Derived class : Scholarship student

- Amount

Derived class : Domestic student

- Province

So, here, every derived class has name, tuition, credits and their own properties. And the properties in the abstract class can be overridden in a derived class if :

```
public abstract int Credits {set;} //now since there is no get accessor in the abstract class.
```

//Now, if the abstract property does not have a get accessor then we can not print out credits in the
//Main() method because no get accessor.

Now let's say, if we add get in the abstract class property :

```
public abstract int Credits {get; set;}
```

//more errors will be shown if the other derived classes doesn't have a get accessor. So rather make it
//virtual property because every abstract class condition must be followed exactly!!

MULTIPLE INHERITANCE : interface

Interfaces are simple classes that only consists of Methods, and any class deriving from an interface must have that method. Also if there id a derived class from a parent class which is from the interface class, then even the derived lass from the parent class must have the methods from the interface class.

Many interfaces can be there too.

Class A:1,2

So that mean s the class A has 2 interfaces.