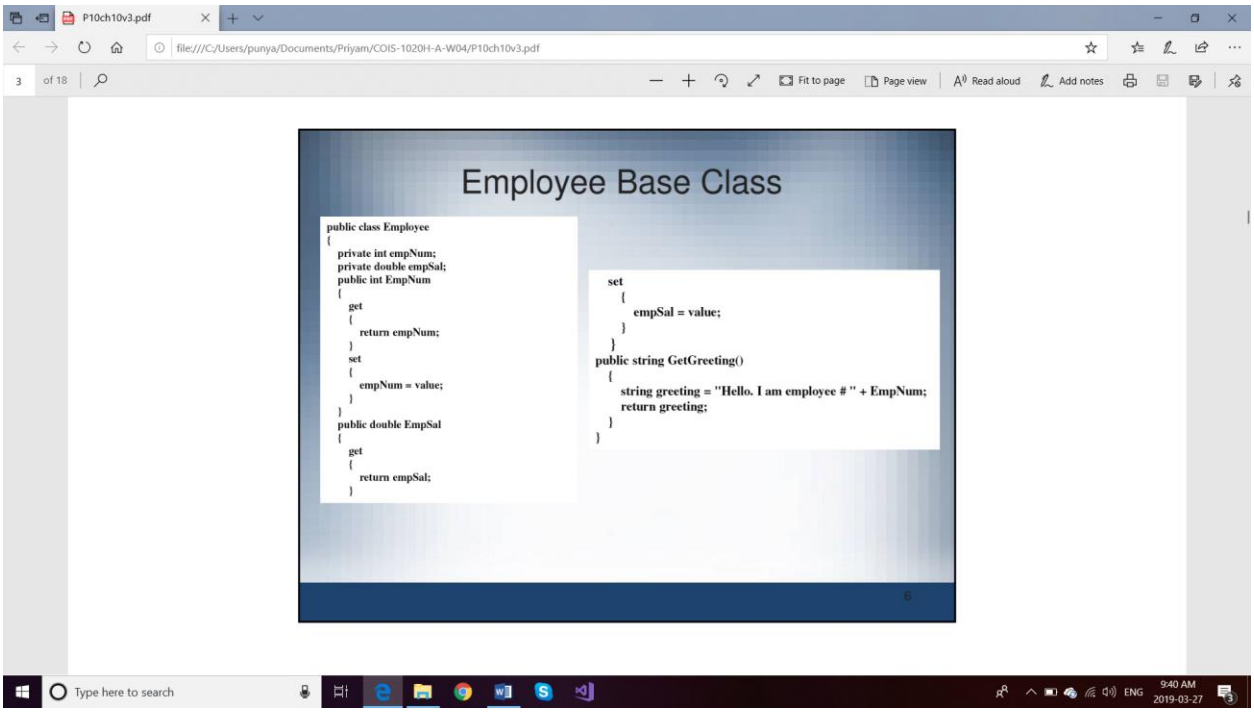
INHERITANCE

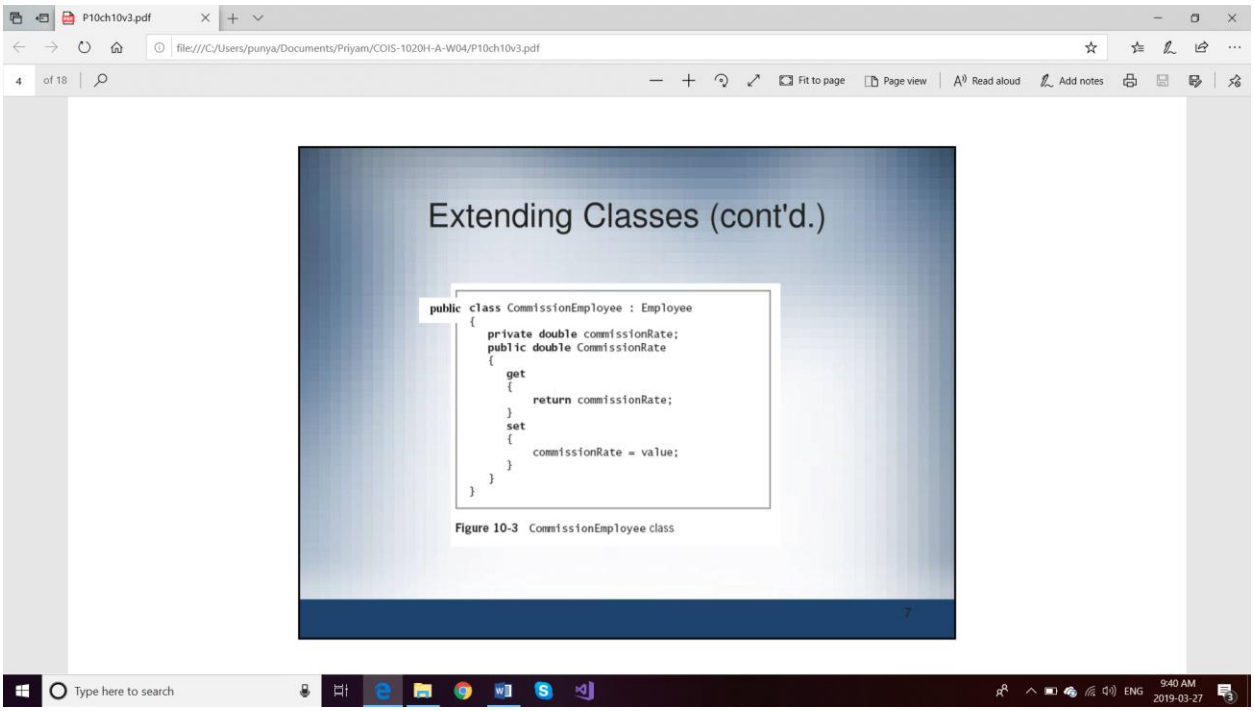
(1) The DERIVED CLASS is the inherited class. And sad as “a truck ‘is a’ vehicle”
(2) Syntax –

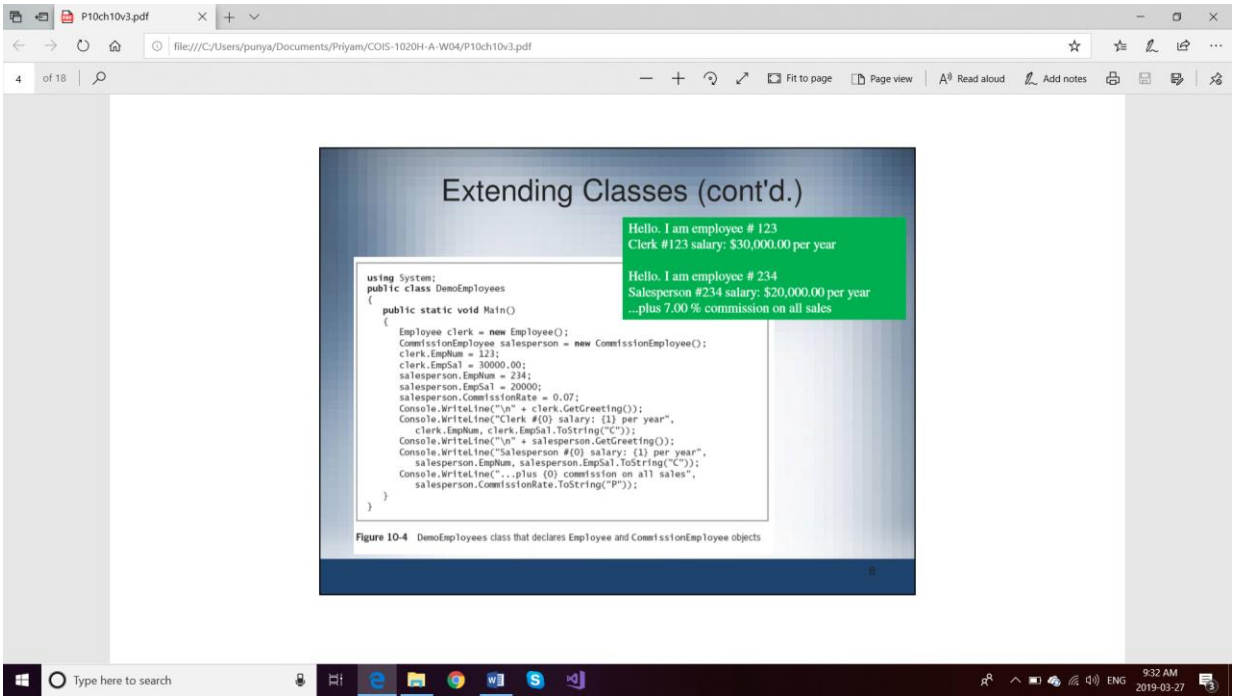
Truck : vehicle

(3) A DERIVED CLASS gets direct access to anything <u>public</u> 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.

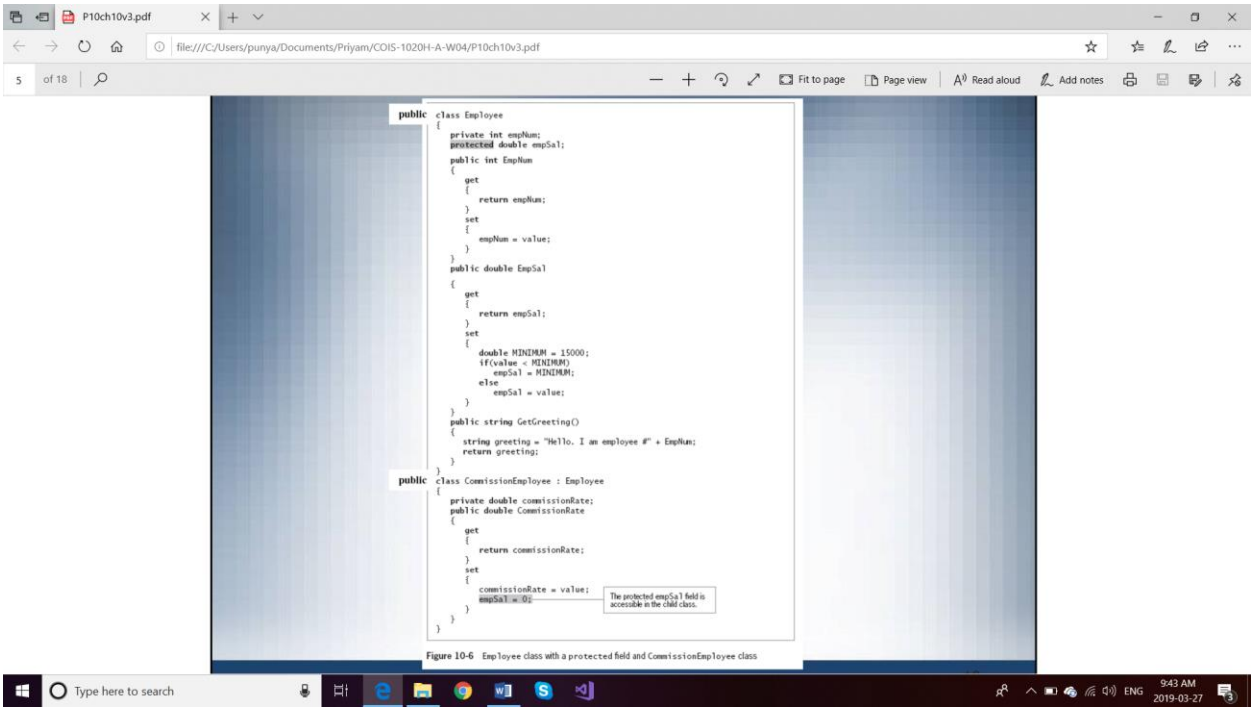
From point of view of Main(), I can assign three different things to the salesperson, because I has access to both Employee class and CommissionEmployee class. And no specific order needs to be followed while assigning values.

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.

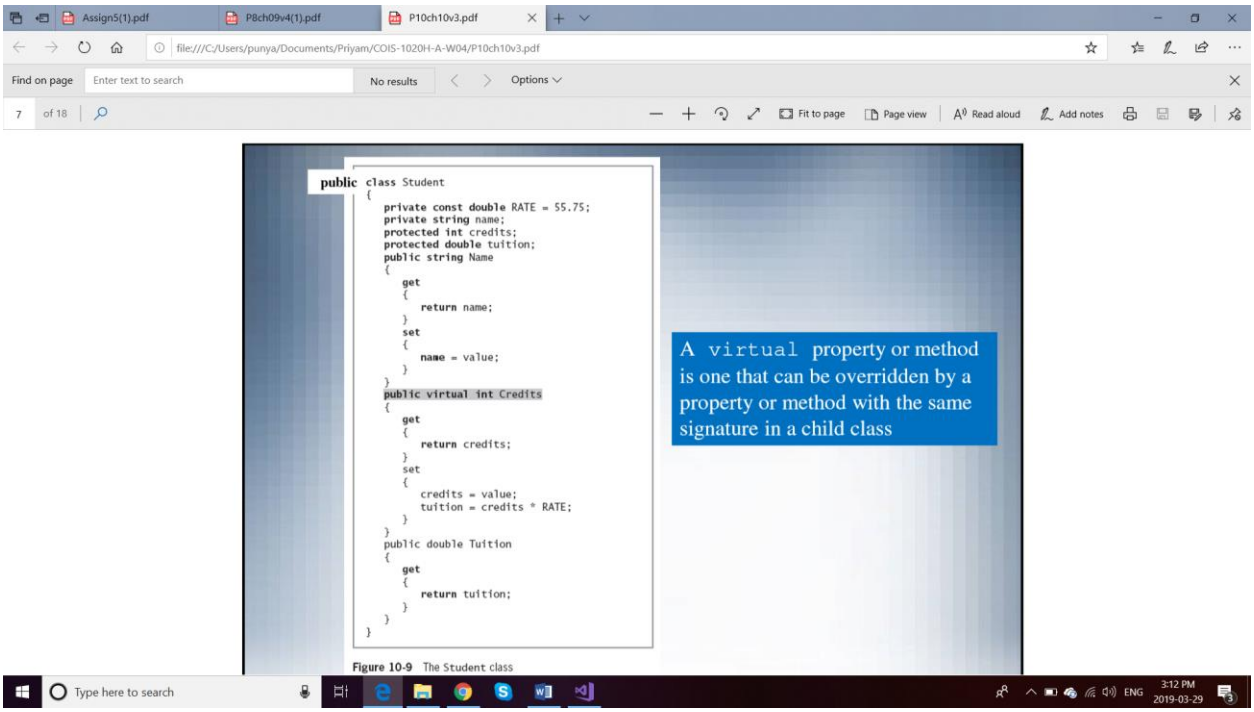
If I do, base.GetGreeting() in the GetGreeting() in the child class, then the control shifts in the base class GetGreeting() method.

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()

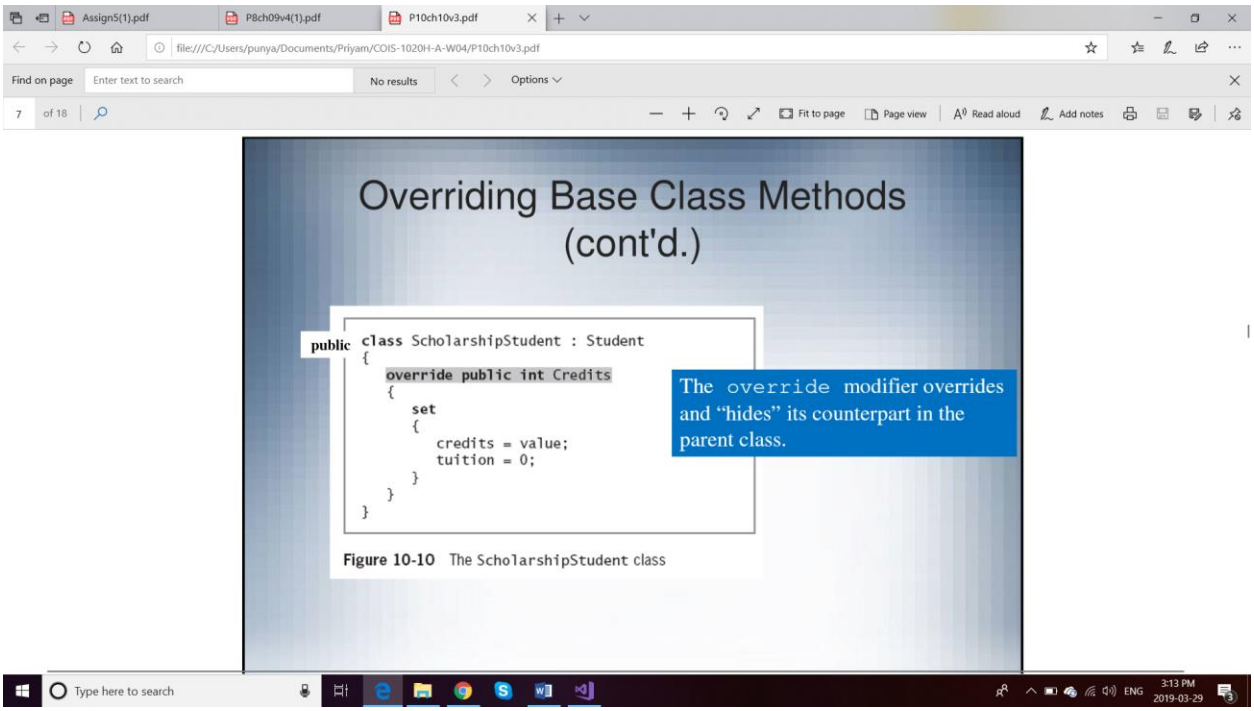
OVERRIDE

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.

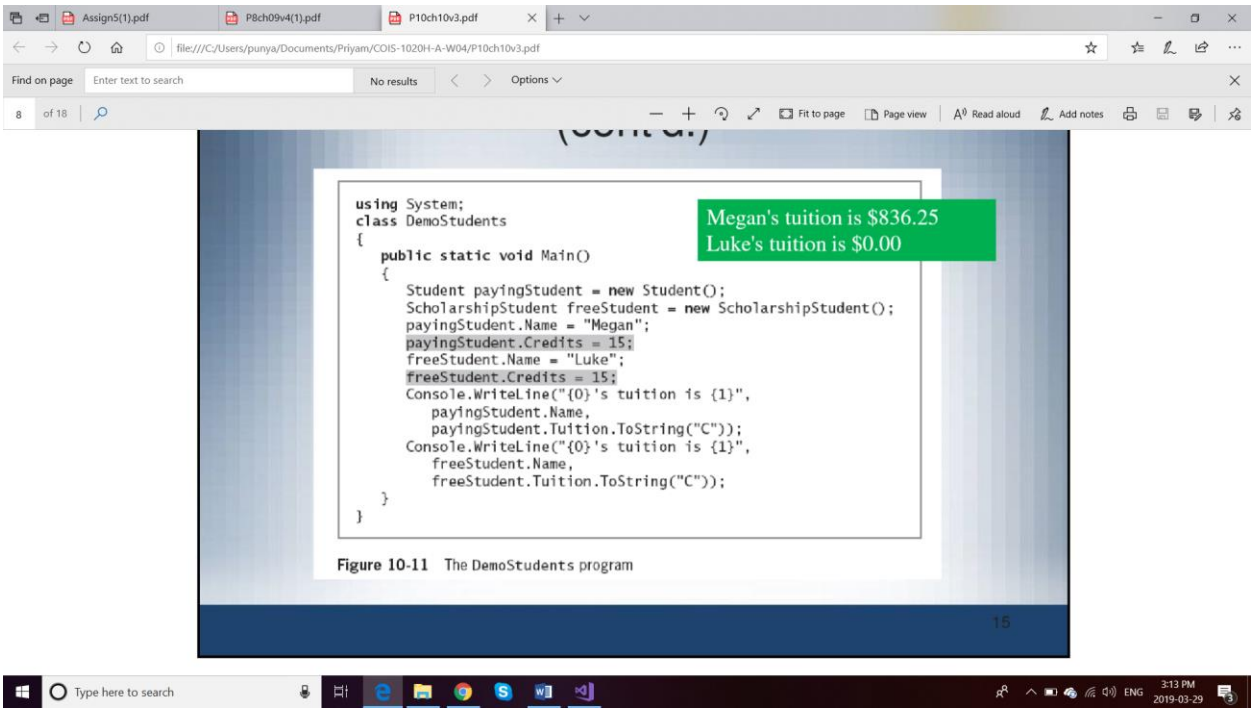
An abstract method HAS to be overridden



This is the base/parent class

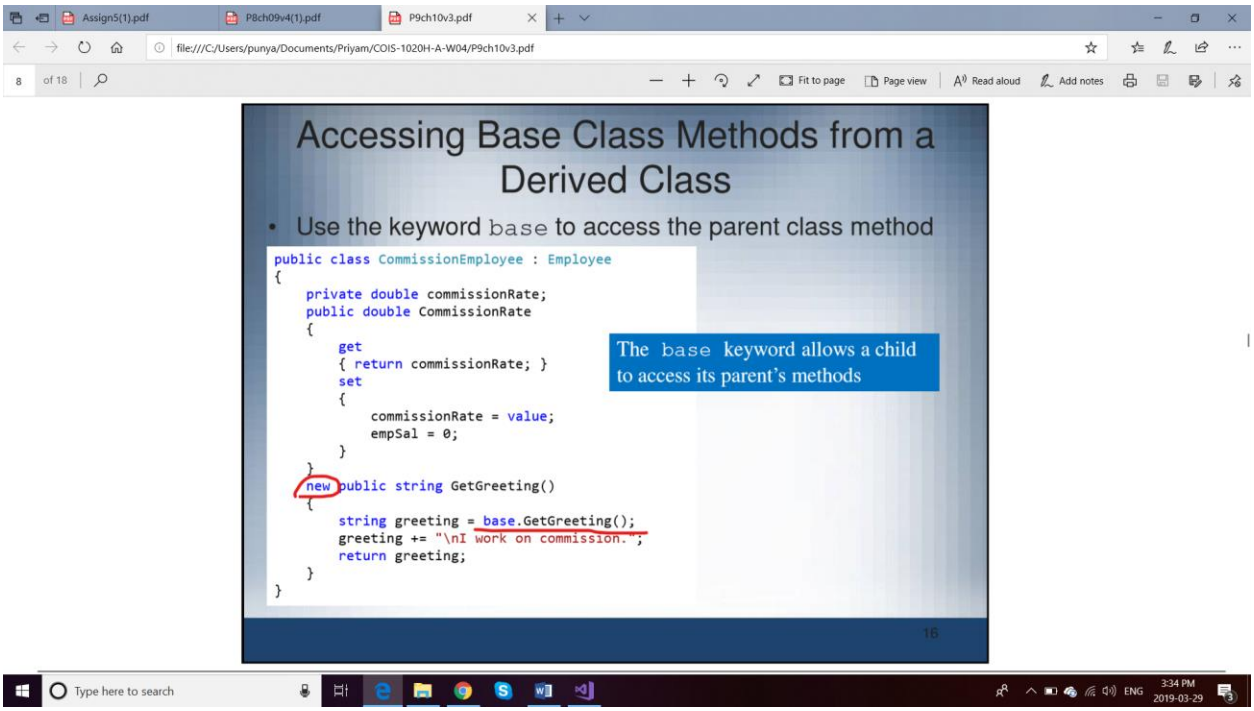


The child class overrides credits.



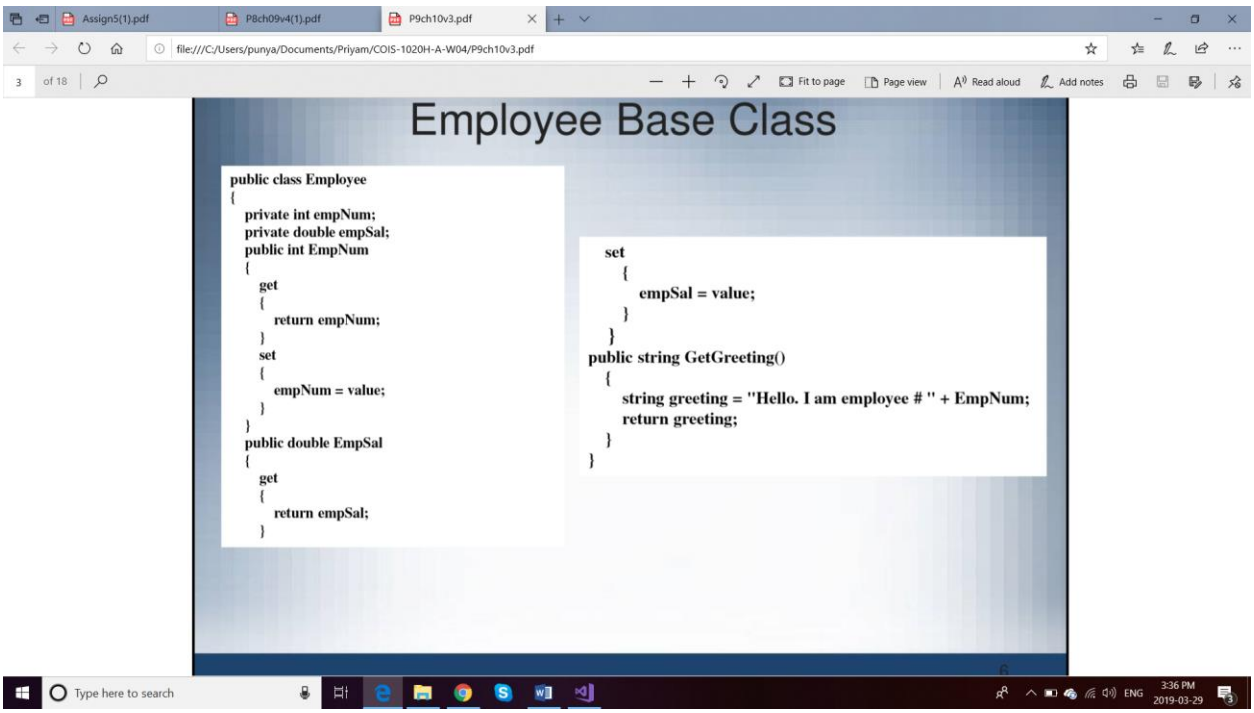
The Main() Method.

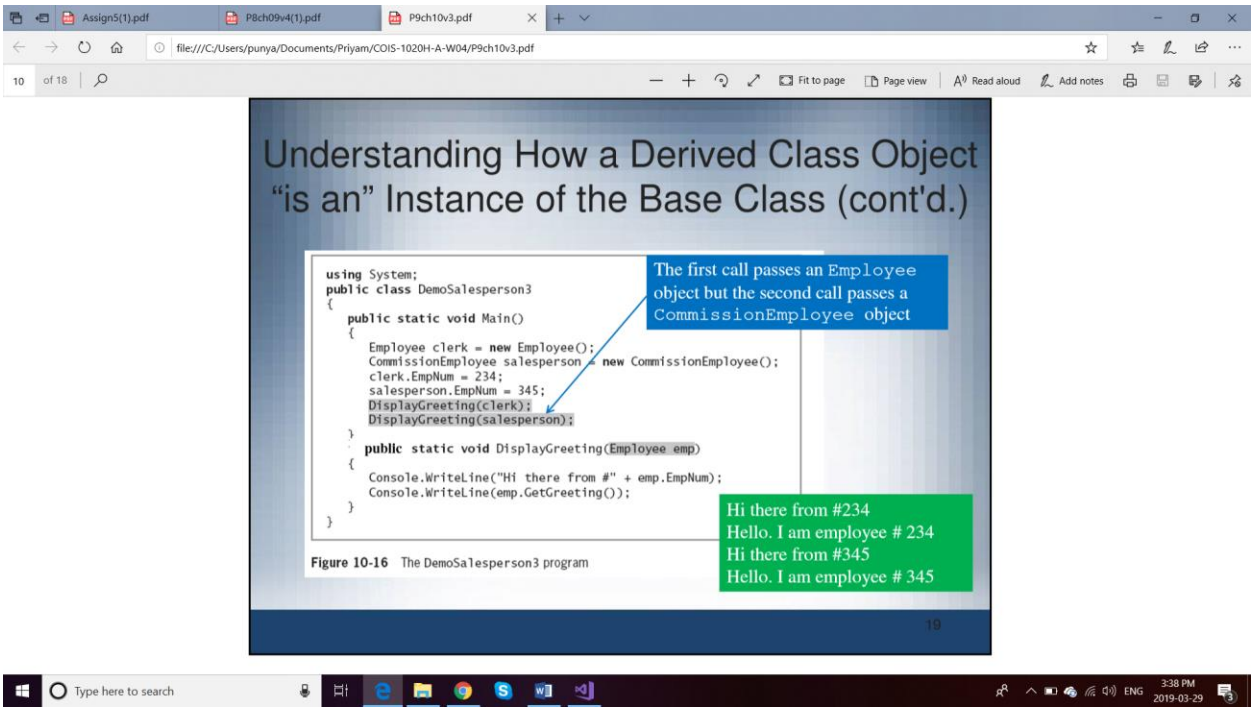
The 'no argument constructors' are being used here.



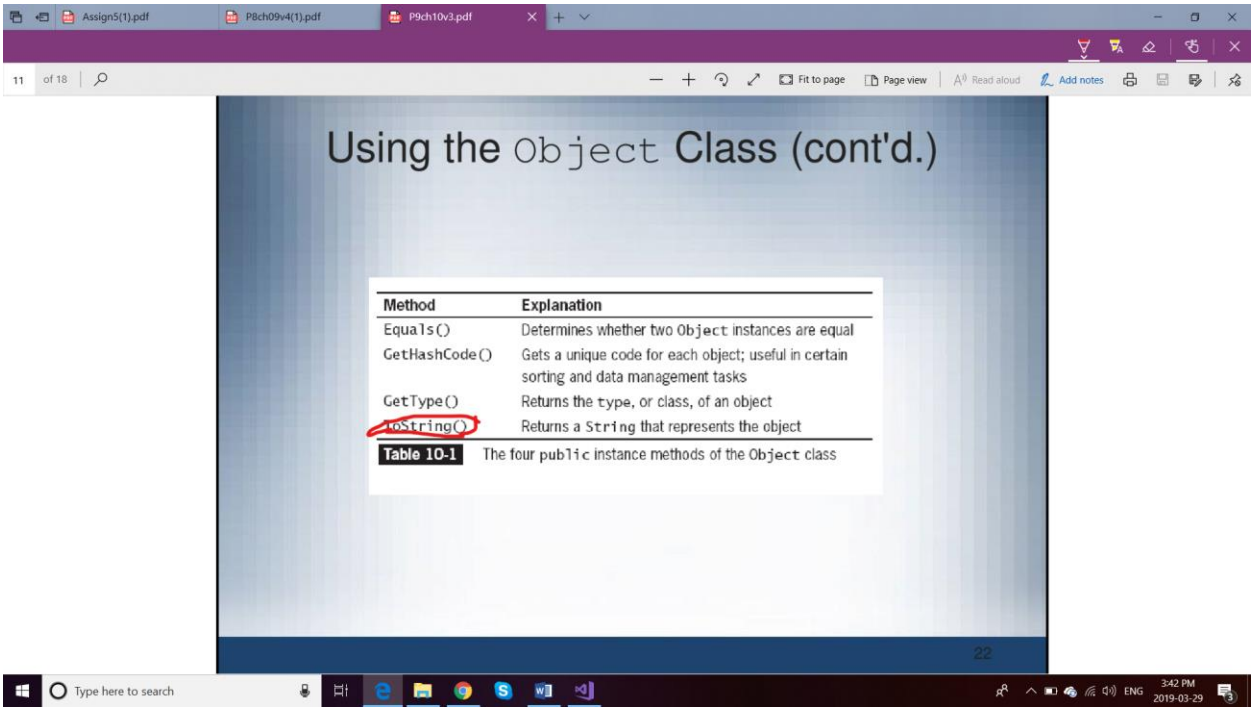
The 'new' keyword has to be used because we used a method with the same name in the base class.

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





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

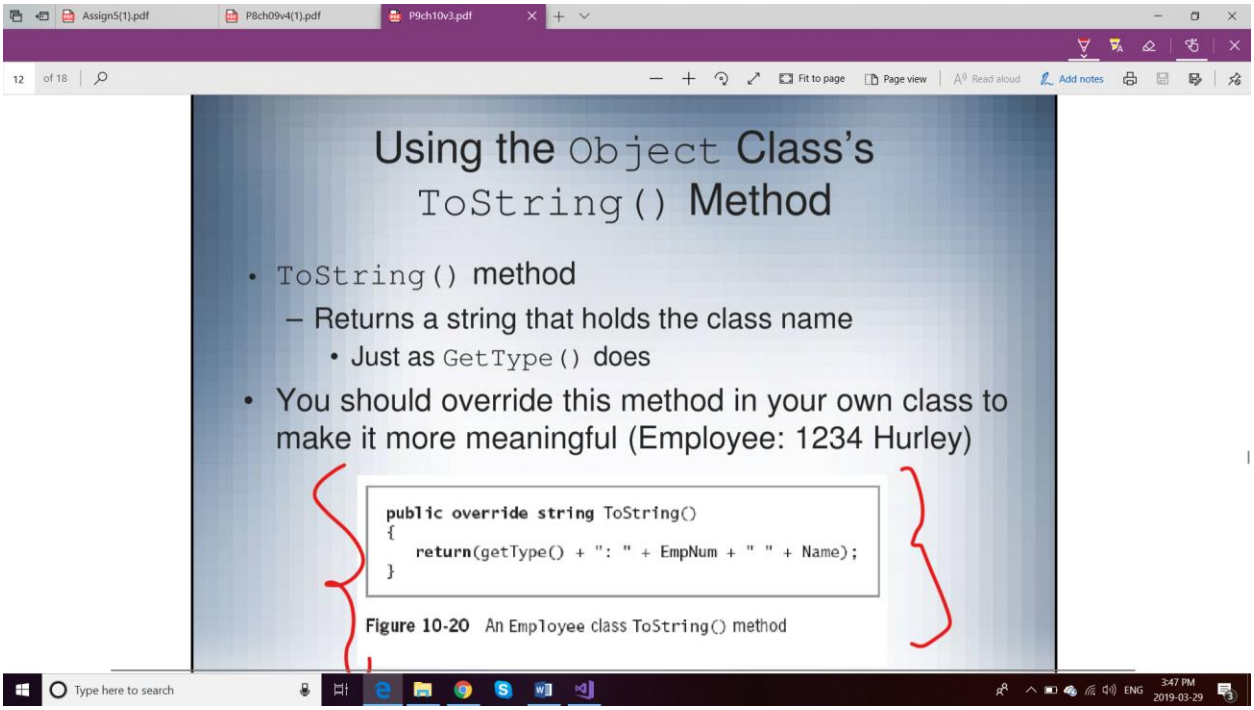


ToString() method in the Object class is automatically called. It turns something into a string.

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.

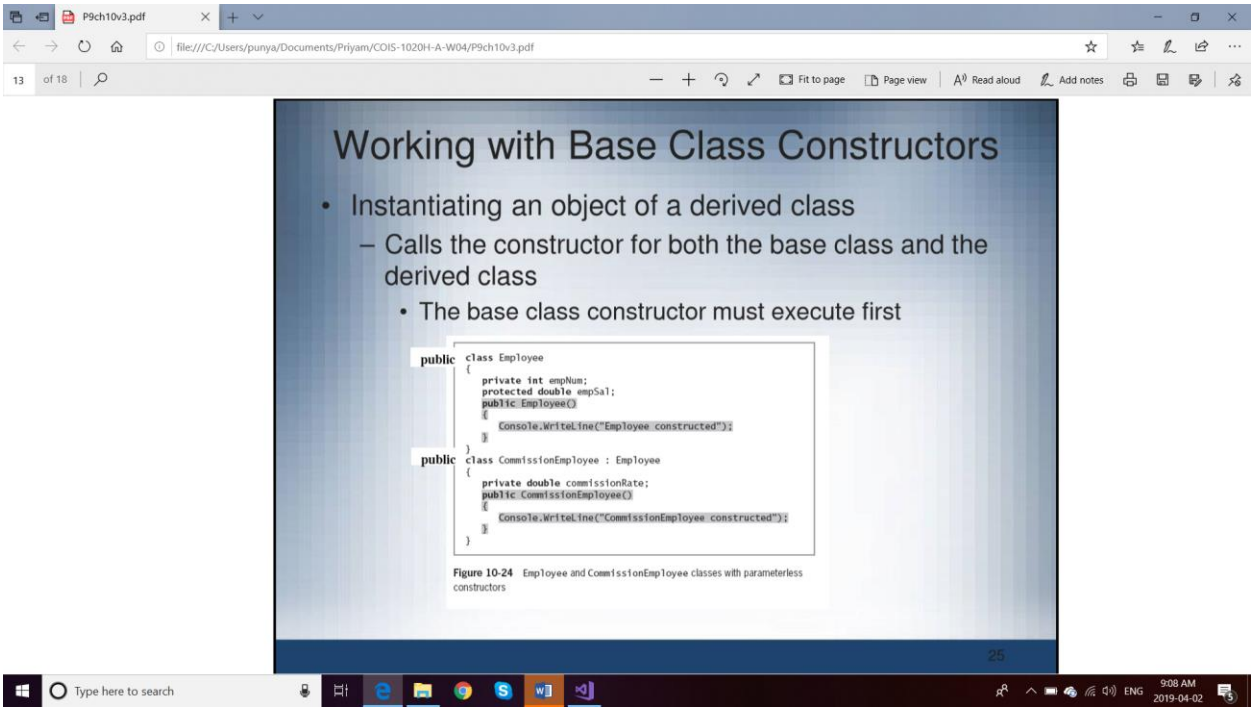


Using the `Object` Class's `ToString()` Method

- `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



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

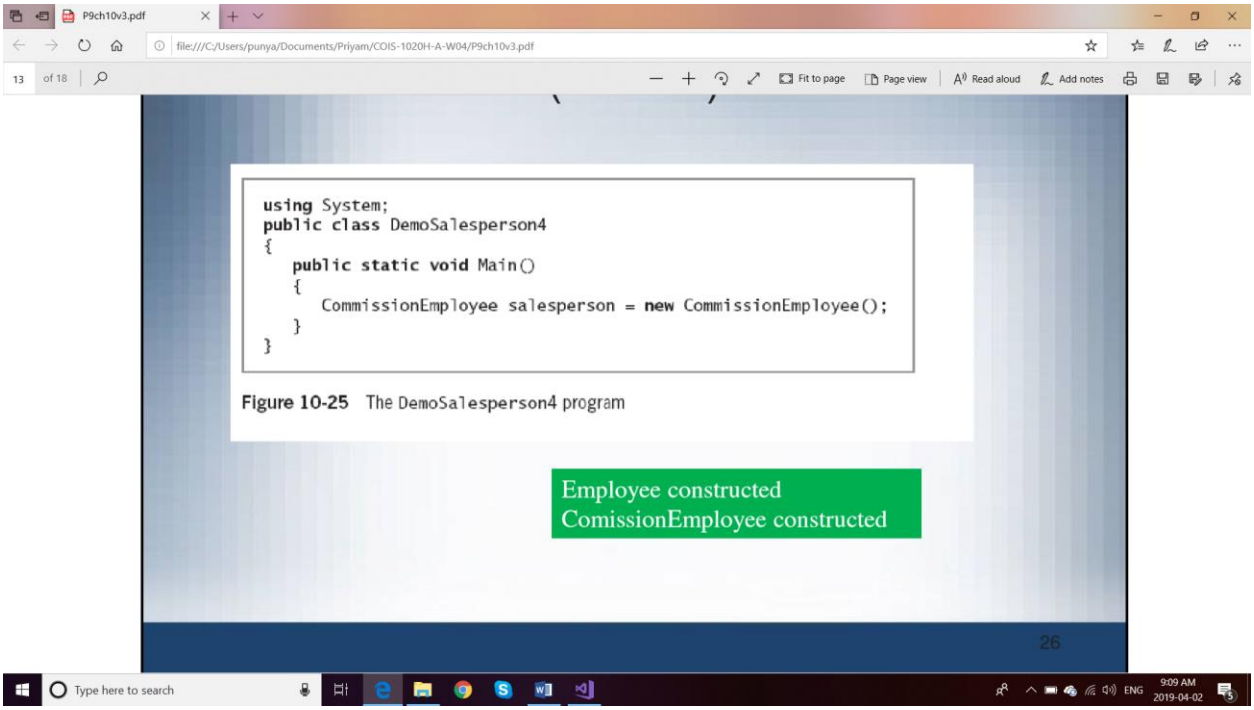
```
public class Employee
{
   private int empNum;
   protected double empSal;
   public Employee()
   {
      Console.WriteLine("Employee constructed");
   }
}
public class CommissionEmployee : Employee
{
   private double commissionRate;
   public CommissionEmployee()
   {
      Console.WriteLine("CommissionEmployee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

Figure 10-24 Employee and CommissionEmployee classes with parameterless constructors

25

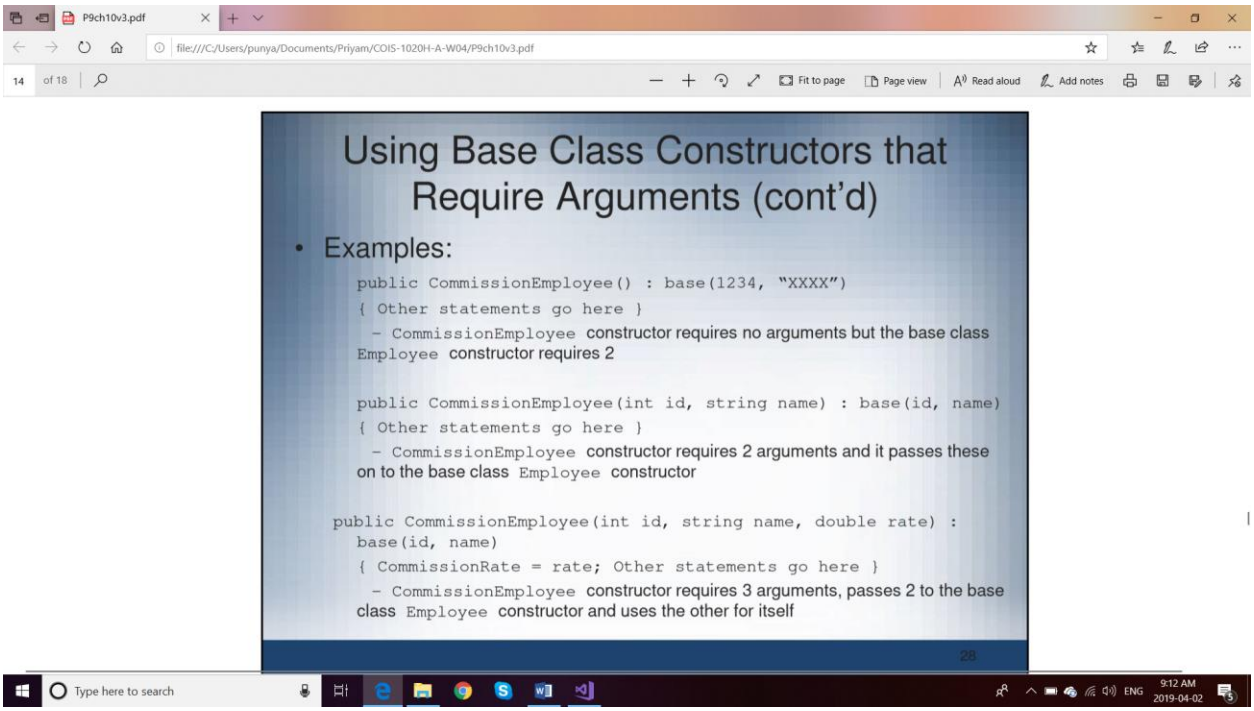
When a child constructor is constructed, first the base constructor is invoked then the child.

That means :



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

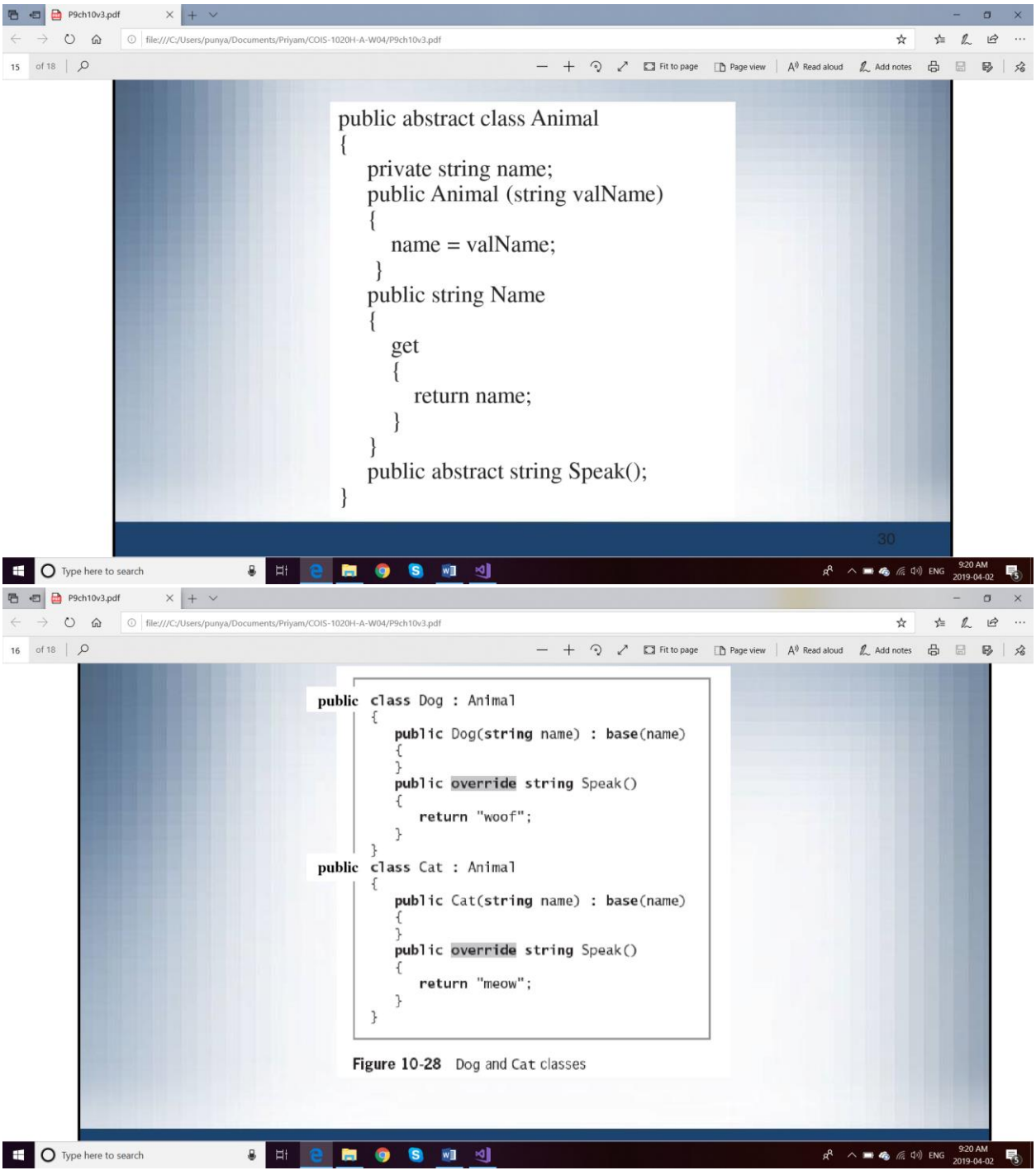
If base constructor is : public Employee(int empId, string empName)



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();
}
```

```
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.