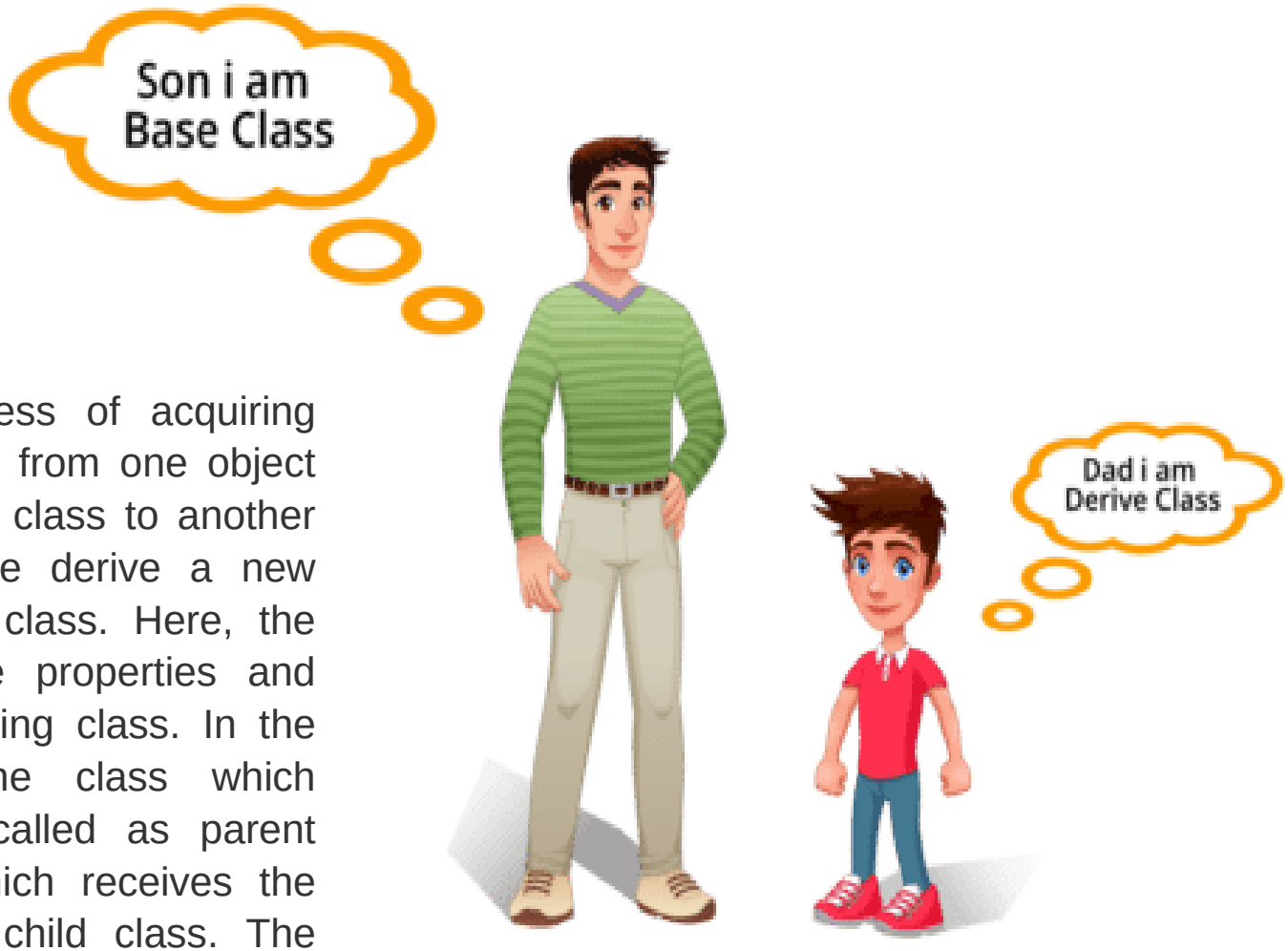
Object Oriented Programming

Inheritance



Inheritance is the process of acquiring properties and behaviors from one object to another object or one class to another class. In inheritance, we derive a new class from the existing class. Here, the new class acquires the properties and behaviors from the existing class. In the inheritance concept, the class which provides properties is called as parent class and the class which receives the properties is called as child class. The parent class is also known as base class or super class. The child class is also known as derived class or sub class.

Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes.

- Allows programmers to create new classes based on an existing class
- Methods and attributes from the parent class are inherited by the newly-created class
- New methods and attributes can be created in the new class, but don't affect the parent class's definition

- It supports the concept of hierarchical classification.

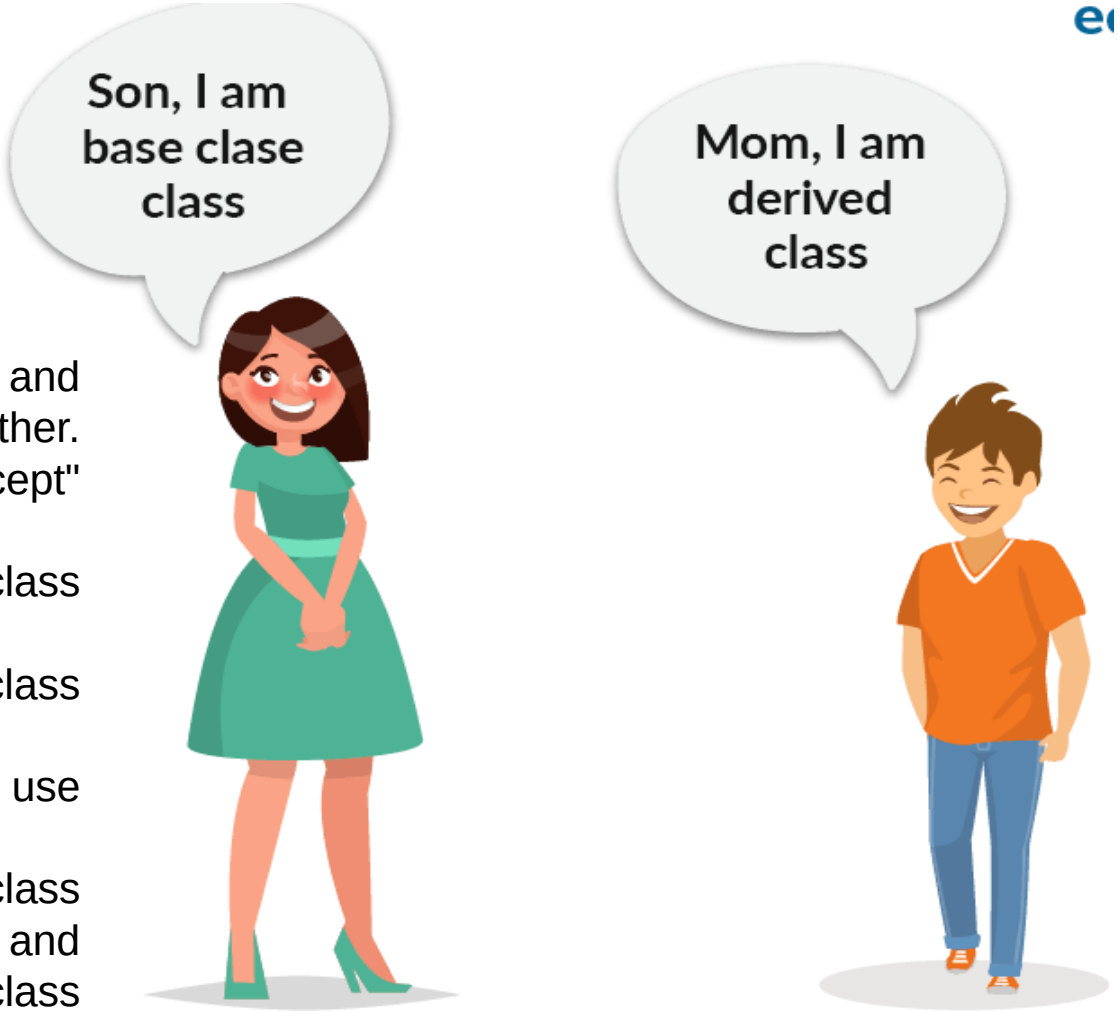
  For example, the bird, 'robin' is a part of class 'flying bird' which is again a part of the class 'bird'.
  The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived

In the inheritance, the properties and behaviors of base class extended to its derived class, but the base class never receive properties or behaviors from its derived class.

A subclass is also called a **derived class** and the class from which it is derived (parent class) is called **superclass** or **base class**.

- **Derived Class** (child) - the class that inherits from another class
- **Base Class** (parent) - the class being inherited from

C# and .NET support *single inheritance* only. That is, a class can only inherit from a single class. However, inheritance is transitive, which allows you to define an inheritance hierarchy for a set of types. In other words, type D can inherit from type C, which inherits from type B, which inherits from the base class type A. Because inheritance is transitive, the members of type A are available to type D.



it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

- **Derived Class** (child) - the class that inherits from another class
- **Base Class** (parent) - the class being inherited from

To inherit from a class, use the : symbol.

In the example below, the Car class (child) inherits the fields and methods from the Vehicle class (parent):

Inheritance

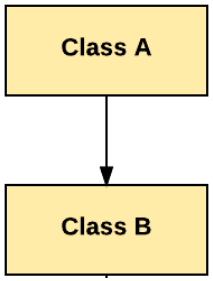
- Inheritance allows child classes to inherit the characteristics of existing parent class
  - Attributes (fields and properties)
  - Operations (methods)
- Child class can extend the parent class
  - Add new fields and methods
  - Redefine methods (modify existing behavior)
- A class can implement an interface by providing implementation for all its methods

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes.

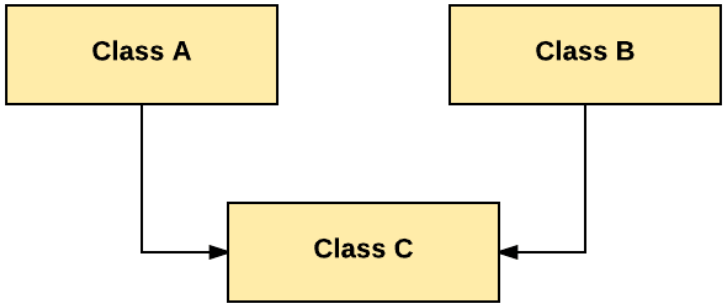
Class hierarchy

The parent-child relationship between classes can be represented in a hierarchical view often called *class tree view*. The class tree view starts with a general class called superclass (sometimes referred to as *base class*, *parent class*, *ancestor class*, *mother class* or *father class*), there are many genealogical metaphors). Derived classes (*child class* or *subclass*) become more specialized further down the tree.

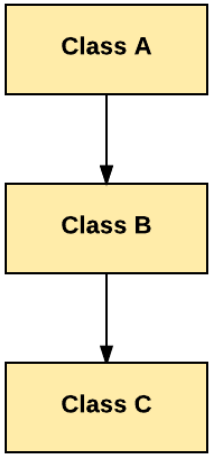
Single Inheritance:
In Single Inheritance one class extends another class (one class only).



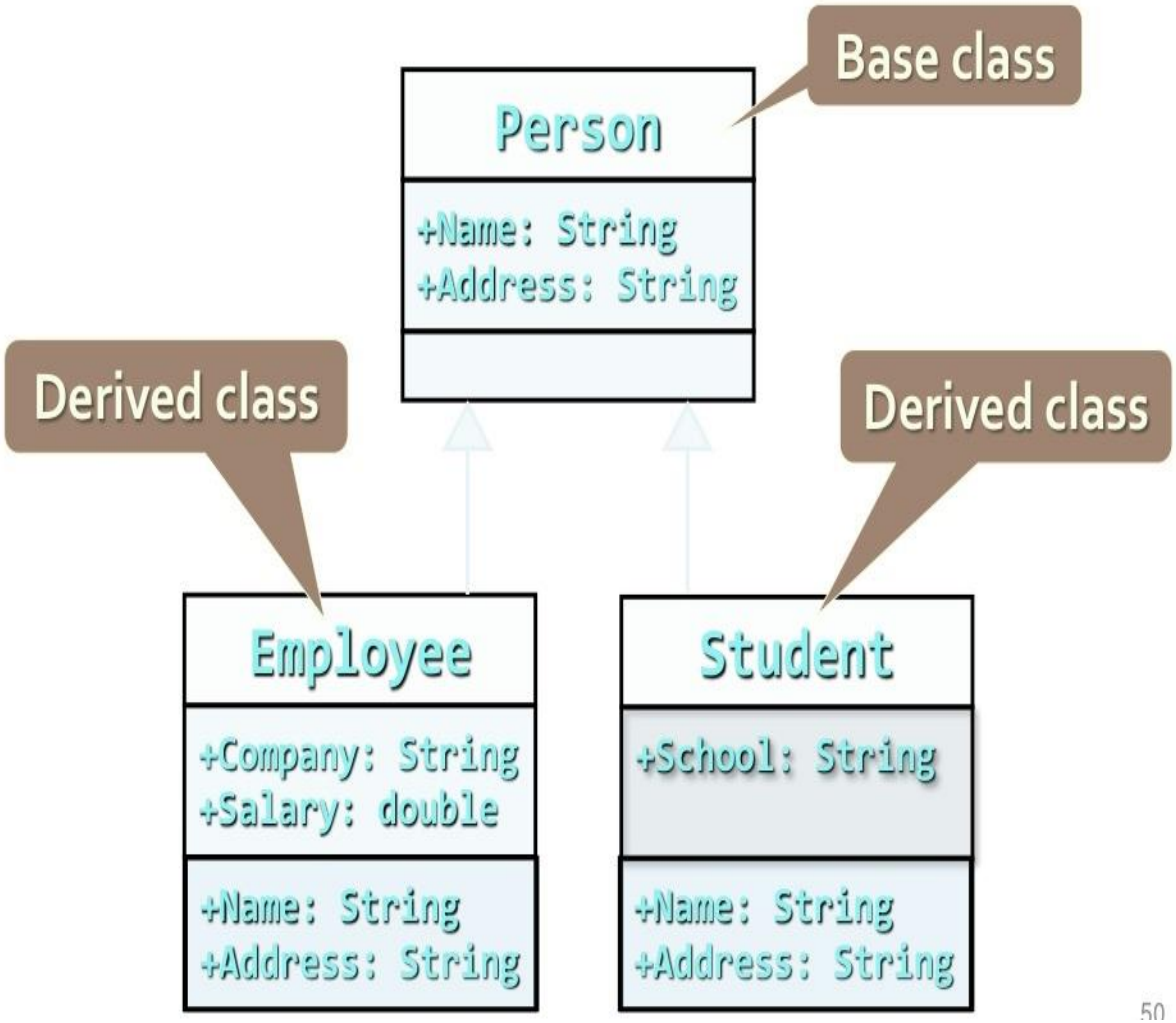
Multiple inheritance
Some object oriented languages, such as C++ allow multiple inheritance, meaning that one class can inherit attributes from two superclasses. This method can be used to group attributes and methods from several classes into one single class.



Hierarchical Inheritance:
In Hierarchical Inheritance, one class is inherited by many sub classes.
Class B, C, and D inherit the same class A.



Inheritance – Example



50

```
class Square
    {
        private double length;

        public double SArea()
        {
            return length * length;
        }

    }
```

```
class Program
    {
        static void Main(string[] args)
        {

            double side;

            side=10;

            Square sq = new Square(side);

            Console.WriteLine(sq.SArea());

            Console.ReadKey();

        }
    }
```

The members (attributes) of the class should be **protected** so they can be accessed within that class or its subclass. It cannot be accessed outside that.

```
class Square
    {
        private double length;

        public double SArea()
        {
            return length * length;
        }

    }
```

```
class circle:square
    {
        private double rad;

        public double CArea()
        {
            return rad * rad * 3.14;
        }

    }
```

```
class Program
    {
        static void Main(string[] args)
        {

            double radius;

            radius=12;

            circle cir = new circle (radius);

            Console.WriteLine(cir.CArea());

            Console.ReadKey();

        }
    }
```

```
class Square
    {
        private double length;

        public double SArea()
        {
            return length * length;
        }

    }
```

```
class circle:square
    {

        public double CArea()
        {
           return length * length * 3.14;
        }

    }
```

```
class Program
    {
        static void Main(string[] args)
        {

            double radius;

            radius=12;

            circle cir = new circle (radius);

            Console.WriteLine(cir.CArea());

            Console.ReadKey();

        }
    }
```

The sealed Keyword

If you don't want other classes to inherit from a class, use the sealed keyword:

If you try to access a sealed class, C# will generate an error

sealed keyword is used to restrict a class from being derived. It means that if we use sealed with a class, we won't be able to create any subclass of it.

We can also use sealed keyword with methods to prevent them for being overridden.

If you don't want other classes to inherit from a class, use the sealed keyword:

Let's look at an example of using sealed with a class.

```
sealed class Vehicle
{
  ...
}

class Car : Vehicle
{
  ...
}
```

```
using System;
sealed class A
{
}
class B: A
{
}
class Test
{
static void Main(string[] args)
{
}
}
```

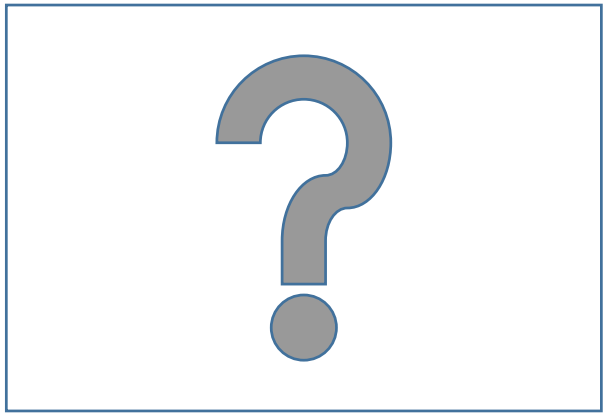
Output

hello.cs(8,7): error CS0509: 'B': cannot derive from sealed type 'A'
In this example, we tried to make a subclass of sealed class *A* and that's why we got the error.

Why And When To Use "Inheritance"?

- It is useful for code reusability: reuse fields and methods of an existing class when you create a new class.

QUESTION



Google Classroom :