Object Oriented Programming (OOP)

Inheritance:

Inheritance enables you to create new classes that reuse, extend, and modify the behavior that is defined in other classes.

The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class.

a derived class is a specialization of the base class.

For example, if you have a base class person, you might have one derived class that is named student and another derived class that is named employee .

A student is an person, and a employee is an person, but each derived class represents different specializations of the base class.

When you define a class to derive from another class, the derived class implicitly gains all the members of the base class, except for its constructors.

The derived class can reuse the code in the base class without having to re-implement it. In the derived class, you can add more members.

The direct base class of a derived class is the base class from which the derived class inherits [via the colon (:) symbol].

With single inheritance, a class is derived from one base class. C# does not support multiple inheritances

An “is-a” relationship represents inheritance. In an “is-a” relationship, an object of a derived class also can be treated as an object of its base class.

A base class's **public** members are accessible anywhere that the program has a reference to an object of that base class or to an object of one of that base class's derived classes.

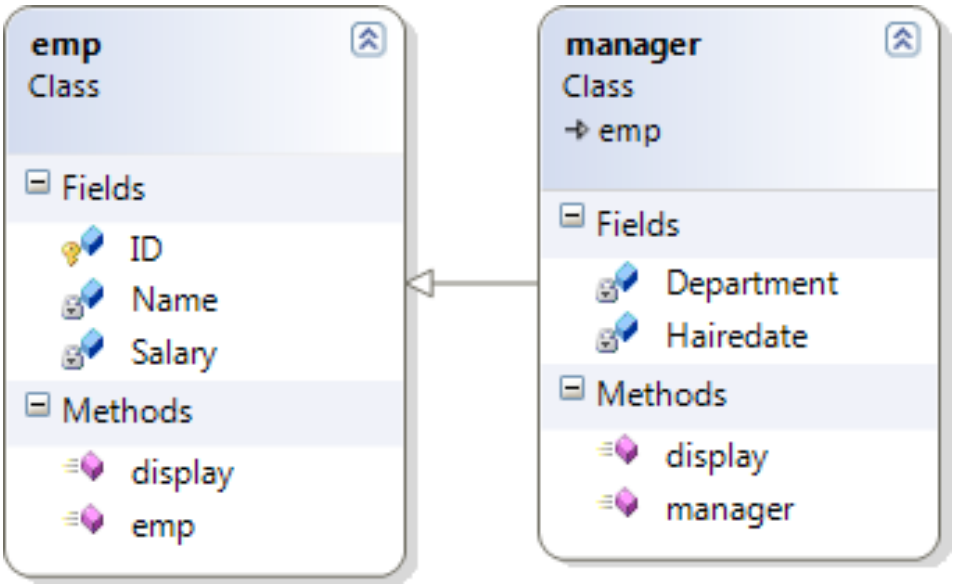
A base class's **private** members are accessible only within the definition of that base class.

A base class's **protected** members have an intermediate level of protection between **public** and **private** access. A base class's **protected** members can be accessed only in that base class or in any classes derived from that base class.

A derived class can redefine a base-class method using the same signature; this is called *overriding* that base-class method.
A base-class method must be declared **virtual** if that method is to be overridden in a derived class.

To override a base-class method definition, a derived class must specify that the derived-class method overrides the base-class method with keyword **override** in the method header.

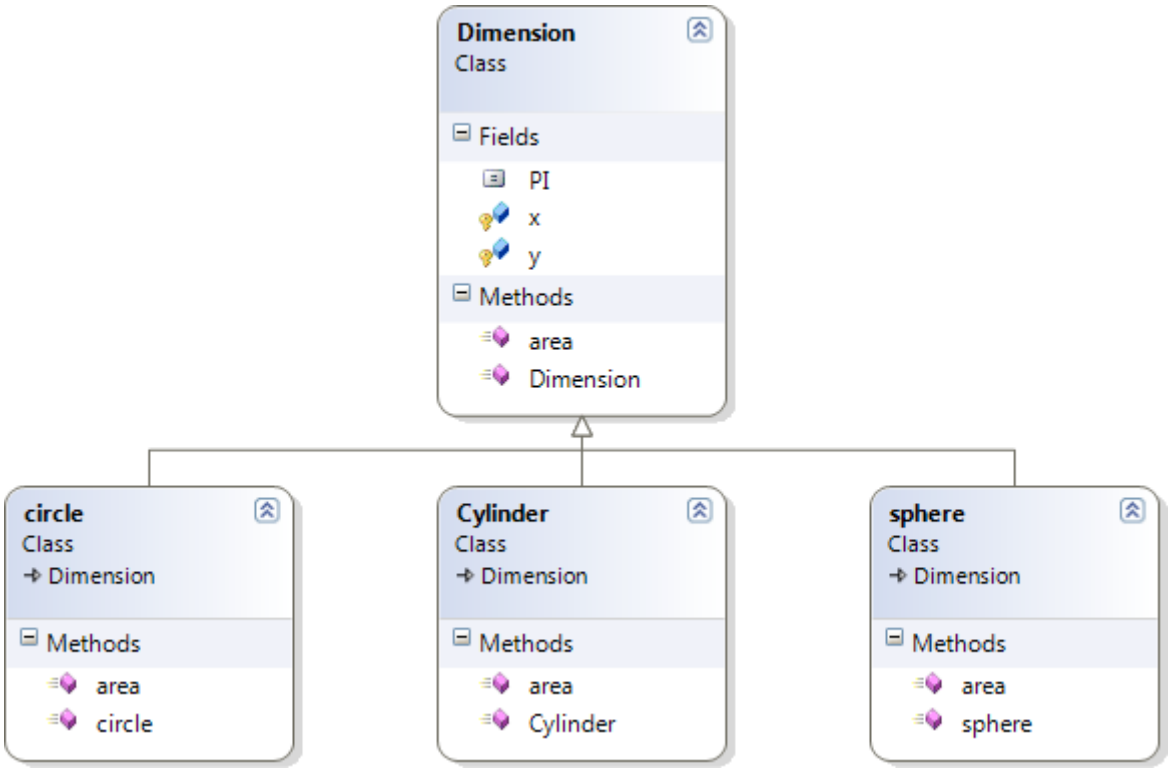
Example:




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Employee1
{
    class emp
    {
        protected int ID;
        string Name;
        double Salary;

        public emp(int id, string n, double s)
        {
            ID = id;
            Name = n;
            Salary = s;
        }

        public virtual void display()
        {
            Console.WriteLine("Emp ID: {0} \nEmp Name: {1} \nEmp Salary: {2}", ID, Name, Salary);
        }
    }

    class manager :emp
    {
        string Department;
        DateTime Hairedate;

        public manager(int id, string n, double s, string d)
            : base(id, n, s)
        {
            Department =  d;
            Hairedate = DateTime.Now;
        }
```

```
        public override void display()
        {
            base.display();
            Console.WriteLine("Department: " + Department+"  \ndate: "+Hairedate);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            manager m = new manager(100, "doaa", 4000, "Cs");
            m.display();
        }
    }
}
```

Example:




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Dimension
{
    class Dimension
    {
        public const double PI = Math.PI;
        protected double x, y;

        public Dimension(double x, double y)
        {
            this.x = x;
            this.y = y;
```

```
        }

        public virtual double area()
        {
            return x * y;
        }
    }

    class circle : Dimension
    {
        public circle(double r)
            : base(r, 0) { }
        public override double area()
        {
            return PI*x*x;
        }
    }

    class sphere :Dimension
    {
        public sphere(double r) : base(r, 0) { }

        public override double area()
        {
            return 4*PI*x*x;
        }
    }
    class Cylinder : Dimension
    {
        public Cylinder(double r, double h) : base(r, h) { }

        public override double area()
        {
            return 2*PI*x*x+2*PI*x*y;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            circle c = new circle(3);
            Console.WriteLine("area of circle {0:f3}",c.area());

            Cylinder cy = new Cylinder(3, 4);
            Console.WriteLine("area of cylinder {0:f3}", cy.area());

            sphere s = new sphere(3);
            Console.WriteLine("area of sphere {0:f3}", s.area());
        }
    }
}
```

Define a class Time with the following attributes, h, m ,and s of types int that describe H

our, Minute, and Second respectively

a) Write a properties for instance variables
b) Write an overloading constructor that initializes h, m, and s attributes
c) Write a method check that check if hour, minute, and second in their range and returntrue and if not in range it will display an error message and return false
d) Write a method Convert to string that display H: M: S mode (AM or PM)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace time
{
class Time
{
      int hour; // 0-23
      int minute; // 0-59
      int second; // 0-59




    public int HOUR
    {
        ge
        t
        {
            return hour;

        }
        set
        {
                if (value>23||value<0)
                {

                Console.WriteLine("wrong value entered\nallowed values are 0-23\nretry to enter hour
again");
                    HOUR = int.Parse(Console.ReadLine());
                }
                else
                {
                    hour = value;
                }
        }
    }
```

```
    public int MINUTE
    {
        ge
        t
        {
            return minute;
        }
        set
        {
                if (value > 59 || value < 0)
                {
                    Console.WriteLine("wrong value entered\nallowed values are 0-59\nretry to
again");                enter minute

                    MINUTE = int.Parse(Console.ReadLine());
                }
                else
                {
                    minute = value;
                }
            }
    }

    public int SECOND
    {
        ge
        t
        {
            return second;

        }
        set
        {
                if (value > 59 || value < 0)
                {
                    Console.WriteLine("wrong value entered\nallowed values are 0-59\nretry to
                    enter second
again");
                    SECOND = int.Parse(Console.ReadLine());
                }
                else
                {
                    second = value;
                }
            }
    }
  public Time()
    {
        HOUR=MINUTE=SECOND = 0;
    }

    public Time(int H, int M, int S)
    {
        HOUR = H; MINUTE = M; SECOND = S;
    }

    public Time(int H, int M)
    {
            HOUR = H; MINUTE = M; SECOND = 0;
        }
```

```
    public Time(int H)
    {
            HOUR = H; MINUTE = SECOND = 0;
        }

    public Time(Time x)
    {
        HOUR = x.HOUR; MINUTE = x.MINUTE; SECOND = x.SECOND;



    Public string ConvertToString()
    {

        string mode,
        op; if (HOUR
        >= 12)
                mode = "PM";
        else
                mode = "AM";
        if (HOUR == 12 || HOUR
            == 0) HOUR = 0;
        else
            HOUR = HOUR % 12;
        op = HOUR.ToString() + " : " + MINUTE.ToString() + " : " + SECOND.ToString()
        + "  " + mode; return op;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Time t1 = new Time(); Console.Write("enter  hour: ");
        t1.HOUR = int.Parse(Console.ReadLine()); Console.Write("enter minute:
        "); t1.MINUTE = int.Parse(Console.ReadLine()); Console.Write("enter
        second: ");
        t1.SECOND = int.Parse(Console.ReadLine()); Console.WriteLine(t1.ConvertToString());
    }
}
}
```

Class Diagram of Time Class: