```
                }

                public int y
                {
                        get   {  return _y;   }
                        set   {  _y = value; }
                }
                public override string ToString()
                {
                        return "This point is  (" + x + ", " + y + ") \n";
                }

        }

public class Circle : Point
        {
                const  double  pi=3.14;
                private double _radius;
                public Circle()  {      }
                public Circle( int X, int Y, double Radius ):base(X,Y)
                {
                        _radius = Radius;
                }
                public double radius
                {
                        get    {   return _radius;    }
                        set
                        {
                                if ( value >= 0 )
                                        _radius = value;
                        }
                }

                public double Diameter()
                {
                        return _radius * 2;
                }

                public double Circumference()
                {
                        return 2*pi*_radius;
                }


                public virtual double Area()
                {
                        return pi*_radius*_radius;
                }

                public override string ToString()
```

```
            {
                    return "In this circle : " + "\n" +
                            "    Center = (" + x + ", " + y + ")"+"\n"+
                            "    Radius = " + radius +"\n"+
                            "    Circumference= "+ Circumference()+"\n" +
                            "    Area= "+ Area() +"\n";
            }
        }

        private void ButCalculate_Click(object sender, EventArgs e)
    {
                    Circle x = new Circle( 3, 5, 10 );
                    MessageBox.Show(x.ToString( ));

        }
```

6-4- Virtual Methods and Override Methods

When an instance method declaration includes a virtual modifier, that method is said to be a virtual method. When no virtual modifier is present, the method is said to be a non- virtual method.

The implementation of a non virtual method is invariant. The implementation is the same whether the method is invoked on an instance of the class in which it is declared or an instance of a derived class. In contrast, the implementation of a virtual method can be superseded by derived classes. The process of superseding the implementation of an inherited virtual method is known as overriding that method.

In a virtual method invocation, the runtime type of the instance for which that invocation takes place determines the actual method implementation to invoke. In a non virtual method invocation, the compile-time type of the instance is the determining factor.

In precise terms, when a method named N is invoked with an argument list A on an instance with a compile-time type C and a runtime type R (where R is either C or a class derived from C), the invocation is processed as follows.

- First, overload resolution is applied to C, N, and A to select a specific method H from the set of methods declared in and inherited by C.

- Then, if H is a nonvirtual method, M is invoked.

- Otherwise, H is a virtual method, and the most derived implementation of H with respect to R is invoked.

For every virtual method declared in or inherited by a class, there exists a most derived implementation of the method with respect to that class. The most derived implementation of a virtual method M with respect to a class R is determined as follows.

- If R contains the introducing virtual declaration of M, then this is the most derived implementation of M.

- Otherwise, if R contains an override of M, then this is the most derived implementation of M.