```
    }
class Dog : Animal
{
   static int _MaxAge = 30;
   string _Type;

   public Dog(string Name, int Age, string Type)
      : base(Name, Age)
   {
      _Type = Type;
   }
   public override void Walk()
   {
      MessageBox.Show("Dog is walking");
   }
   public new void Talk()
   {
      MessageBox.Show(" Dog is barking ");
   }
    }

   private void ButCalculate_Click(object sender, EventArgs e)
   {
      Dog x = new Dog("Doi",40,"Sharlot");
      x.Walk();
      x.Talk();
       }
```

5-2-1- Private Members and protected Members

With inheritance, private members of a base class are not accessible directly from that class's derived classes, but these private base-class members are still inherited. All other base-class members retain their original member access when they become members of the derived class (e.g., public members of the base class become public members of the derived class, and, as we will soon see, protected members of the base class become protected members of the derived class). Through these inherited base-class members, the derived class can manipulate private members of the base class (if these inherited members provide such functionality in the base class).

Using protected access offers 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.

The use of protected variables allows for a slight increase in performance, because we avoid incurring the overhead of a method call to a property's set or get accessor. However, in most C# applications, in which user interaction comprises a large part of the execution time, the optimization offered through the use of protected variables is negligible.

Using protected instance variables creates two major problems:

First, the derived class object does not have to use a property to set the value of the base-class's protected data. Therefore, a derived-

class object can easily assign an illegal value to the protected data, thus leaving the object in an inconsistent state.

The second problem with using protected data is that derived-class methods are more likely to be written to depend on base-class implementation. In practice, derived classes should depend only on the base-class services (i.e., non-private methods and properties) and not on base-class implementation. With protected data in the base class, if the base-class implementation changes, we may need to modify all derived classes of that base class. In such a case, the software is said to be fragile or brittle. The programmer should be able to change the base-class implementation freely, while still providing the same services to derived classes.

Derived-class methods normally can refer to public, protected and internal members of the base class simply by using the member names. When a derived-class method overrides a base-class member, the base-class member can be accessed from the derived class by preceding the base-class member name with keyword base, followed by the dot operator.

6-3- Structural inheritance

In this section, we use a point-circle hierarchy to discuss the relationship between a base class and a derived class. We divide our discussion of the point-circle relationship into several parts. First, we create class point, which directly inherits from class System.Object and contains as private data an x-y coordinate pair. Then, we create class Circle, which directly inherits from class Point. The point-circle relationship may seem unnatural when we discuss it in the context of a circle “is a” point. This example teaches what is sometimes called structural inheritance; the example focuses on the “mechanics” of inheritance and how a base class and a derived class relate to one another.

Example6_3_1:

```
public class Point
        {
                 int _x, _y;
                public Point()
                {

                }
                public Point( int X, int Y )
                {
                        x = X;
                        y = Y;
                }

                public int x
                {
                        get   {  return _x;   }
                        set   {  _x = value; }
```