- Otherwise, the most derived implementation of M with respect to R is the same as the most derived implementation of M with respect to the direct base class of R.

The following example illustrates the differences between virtual and nonvirtual methods.

```
class A
{
        public void F(){MessageBox.Show("A.F");}
        public virtual void G(){MessageBox.Show("A.G");}
}
class B:A
{
        new public void F(){MessageBox.Show("B.F");}
        public override void G(){MessageBox.Show("B.G");}
}

    private void ButCalculate_Click(object sender, EventArgs e)
    {

                B b = new B();
                A a =b;
                a.F();
                b.F();
                a.G();
                b.G();

}
```

In the example, A introduces a nonvirtual method F and a virtual method a. The class B introduces a new nonvirtual method F, thus hiding the inherited F, and also overrides the inherited method a.

Notice that the statement a.G() invokes B.G, not A.G. This is because the runtime type of the instance (which is B), not the compile-time type of the instance (which is A), determines the actual method implementation to invoke.

Because methods are allowed to hide inherited methods, it is possible for a class to contain several virtual methods with the same signature. This does not present an ambiguity problem because all but the most derived method are hidden. In the example

```
class A
{
        public virtual void F(){MessageBox.Show("A.F");}
}
class B:A
{
        public override void F(){MessageBox.Show("B.F");}
}
class C:B
```

```
{
        new public virtual void F(){MessageBox.Show("C.F");}
}
class D:C
{
        public override void F(){MessageBox.Show("D.F");}
}
    private void ButCalculate_Click(object sender, EventArgs e)
    {
            D d = new D();
            A a = d;
            B b = d;
            C c = d;
            a.F();
            b.F();
            c.F();
            d.F();
        }
```

The C and D classes contain two virtual methods with the same signature: the one introduced by A and the one introduced by C. The method introduced by C hides the method inherited from A. Thus, the override declaration in D overrides the method introduced by C, and it is not possible for D to override the method introduced by A.

Note that it is possible to invoke the hidden virtual method by accessing an instance of D through a less derived type in which the method is not hidden.

5-4-1- Override Methods

When an instance method declaration includes an override modifier, the method is said to be an override method. An override method overrides an inherited virtual method with the same signature. Whereas a virtual method declaration introduces a new method, an override method declaration specializes an existing inherited virtual method by providing a new implementation of that method.

The method overridden by an override declaration is known as the overridden base method. For an override method M declared in a class C, the overridden base method is determined by examining each base class of C, starting with the direct base class of c and continuing with each successive direct base class until an accessible method with the same signature as M is located. For the purposes of locating the overridden base method, a method is considered accessible if it is public, if it is protected, if it is protected internal, or if it is internal and declared in the same program as C.

A compile-time error occurs unless all of the following are true for an override declaration.

- An overridden base method can be located as described previously.

- The overridden base method is a virtual, abstract, or override method. In other words, the overridden base method cannot be static or nonvirtual.