

B E G I N N E R S ' S
C # T U T O R I A L
8 . C L A S S I N H E R I T A N C E

WRITTEN BY JOE MAYO
JMAYO@CSHARP-STATION.COM
UPDATED 09/09/00, 08/12/01, 12/03/03

CONVERTED TO PDF BY ASH WEAVER 02/09/03

WWW.CSHARP-STATION.COM

This lesson teaches about C# Inheritance. Our objectives are as follows:

- Implement Base Classes.
- Implement Derived Classes.
- Initialize Base Classes from Derived Classes.
- Learn How to Call Base Class Members.
- Learn How to Hide Base Class Members.

Inheritance is one of the primary concepts of object-oriented programming. It allows you to reuse existing code. Through effective employment of reuse, you can save time in your programming.

Listing 8-1. Inheritance: BaseClass.cs

```
using System;

public class ParentClass
{
    public ParentClass()
    {
        Console.WriteLine("Parent Constructor.");
    }

    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}

public class ChildClass : ParentClass
{
    public ChildClass()
    {
        Console.WriteLine("Child Constructor.");
    }

    public static void Main()
    {
        ChildClass child = new ChildClass();

        child.print();
    }
}
```

Output:

```
Parent Constructor.
Child Constructor.
I'm a Parent Class.
```

Listing 8-1 shows two classes. The top class is named *ParentClass* and the main class is called *ChildClass*. What we want to do is create a child class, using existing code from *ParentClass*.

First we must declare our intention to use *ParentClass* as the base class of *ChildClass*. This is accomplished through the *ChildClass* declaration `public class ChildClass : ParentClass`. The base class is specified by adding a colon, ":", after the derived class identifier and then specifying the base class name.

C# supports single class inheritance only. Therefore, you can specify only one base class to inherit from. However, it does allow multiple *interface* inheritance, a subject covered in a later lesson.

ChildClass has exactly the same capabilities as *ParentClass*. Because of this, you can also say *ChildClass* "is" a *ParentClass*. This is shown in the *Main()* method of *ChildClass* when the *print()* method is called. *ChildClass* does not have its own *print()* method, so it uses the *ParentClass print()* method. You can see the results in the 3rd line of output.

Base classes are automatically instantiated before derived classes. Notice the output from Listing 8-1. The *ParentClass* constructor executed before the *ChildClass* constructor.

Listing 8-2. Derived Class Communicating with Base Class: BaseTalk.cs

```
using System;

public class Parent
{
    string parentString;
    public Parent()
    {
        Console.WriteLine("Parent Constructor.");
    }
    public Parent(string myString)
    {
        parentString = myString;
        Console.WriteLine(parentString);
    }
    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}

public class Child : Parent
{
    public Child() : base("From Derived")
    {
        Console.WriteLine("Child Constructor.");
    }
    public new void print()
    {
        base.print();
        Console.WriteLine("I'm a Child Class.");
    }
    public static void Main()
    {

```

```
    Child child = new Child();  
    child.print();  
    ((Parent)child).print();  
} }
```

Output:
From Derived

Child Constructor.

I'm a Parent Class.

I'm a Child Class.

I'm a Parent Class.

Derived classes can communicate with base classes during instantiation. Listing 8-2 shows how this is done at the child constructor declaration. The colon, ":", and keyword *base* call the base class constructor with the matching parameter list. The first line of output shows the base class constructor being called with the *string* "From Derived".

Sometimes you may want to create your own implementation of a method that exists in a base class. The *Child* class does this by declaring its own *print()* method. The *Child print()* method hides the *Parent print()* method. The effect is the *Parent print()* method will not be called, unless we do something special to make sure it is called.

Inside the *Child print()* method, we explicitly call the *Parent print()* method. This is done by prefixing the method name with "*base.*". Using the *base* keyword, you can access any of a base class *public* or *protected* class members. The output from the *Child print()* method is on output lines 3 and 4.

Another way to access base class members is through an explicit cast. This is done in the last statement of the *Child* class *Main()* method. Remember that a derived class is a specialization of its base class. This fact allows us to perform a cast on the derived class, making it an instance of its base class. The last line of output from Listing 8-2 shows the *Parent print()* method was indeed executed.

Notice the *new* modifier on the *Child* class *print()* method. This enables this method to hide the *Parent* class *print()* method, thus explicitly preventing polymorphism. Without the *new* modifier, the compiler will produce a warning to draw your attention to this. See the next lesson for a detailed discussion of polymorphism.

In summary, you know how to create a derived/base class relationship. You can control instantiation of your base class and call its methods either implicitly or explicitly. You also understand that a derived class is a specialization of its base class.