**The C# Station Tutorial**

**by**[**Joe Mayo**](http://twitter.com/#!/JoeMayo)**http://www.csharp-station.com/Tutorial/CSharp/**

## Lesson 9: Polymorphism

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

* Learn What Polymorphism Is.
* Implement a Virtual Method.
* Override a Virtual Method.
* Use Polymorphism in a Program.

Another primary concept of object-oriented programming is Polymorphism. It allows you to invoke derived class methods through a base class reference during run-time. This is handy when you need to assign a group of objects to an array and then invoke each of their methods. They won't necessarily have to be the same object type. However, if they're related by inheritance, you can add them to the array as the inherited type. Then if they all share the same method name, that method of each object can be invoked. This lesson will show you how to accomplish this.

##### Listing 9-1. A Base Class With a Virtual Method: DrawingObject.cs

using System;  
  
public class DrawingObject  
{  
    public virtual void Draw()  
    {  
        Console.WriteLine("I'm just a generic drawing object.");  
    }  
}

Listing 9-1 shows the DrawingObject class. This will be the base class for other objects to inherit from. It has a single method namedDraw(). The Draw() method has a virtual modifier. The virtual modifier indicates to derived classes that they can override this method. The Draw() method of the DrawingObject class performs a single action of printing the statement, "I'm just a generic drawing object.", to the console.

##### Listing 9-2. Derived Classes With Override Methods: Line.cs, Circle.cs, and Square.cs

using System;  
  
public class Line : DrawingObject  
{  
    public override void Draw()  
    {  
        Console.WriteLine("I'm a Line.");  
    }  
}  
  
public class Circle : DrawingObject  
{  
    public override void Draw()  
    {  
        Console.WriteLine("I'm a Circle.");  
    }  
}  
  
public class Square : DrawingObject  
{  
    public override void Draw()  
    {  
        Console.WriteLine("I'm a Square.");  
    }  
}

Listing 9-2 shows three classes. These classes inherit the DrawingObject class. Each class has a Draw() method and each Draw()method has an override modifier. The override modifier allows a method to override the virtual method of its base class at run-time. Theoverride will happen only if the class is referenced through a base class reference. Overriding methods must have the same signature, name and parameters, as the virtual base class method it is overriding.

##### Listing 9-3. Program Implementing Polymorphism: DrawDemo.cs

using System;  
  
public class DrawDemo  
{  
    public static int Main( )  
    {  
        DrawingObject[] dObj = new DrawingObject[4];  
  
        dObj[0] = new Line();  
        dObj[1] = new Circle();  
        dObj[2] = new Square();  
        dObj[3] = new DrawingObject();  
  
        foreach (DrawingObject drawObj in dObj)  
        {  
            drawObj.Draw();  
        }  
  
        return 0;  
    }  
}

Listing 9-3 shows a program that uses the classes defined in Listing 9-1 and Listing 9-2. This program implements polymorphism. In the Main() method of the DrawDemo class, there is an array being created. The type of object in this array is the DrawingObject class. The array is named dObj and is being initialized to hold four objects of type DrawingObject.

Next the dObj array is initialized. Because of their inheritance relationship with the DrawingObject class, the Line, Circle, and Squareclasses can be assigned to the dObj array. Without this capability, you would have to create an array for each type. Inheritance allows derived objects to act like their base class, which saves work.

After the array is initialized, there is a foreach loop that looks at each element of the array. Within the foreach loop the Draw() method is invoked on each element of the dObj array. Because of polymorphism, the run-time type of each object is invoked. The type of the reference object from the dObj array is a DrawingObject. However, that doesn't matter because the derived classes override the virtualDraw() method of the DrawingObject class. This makes the overriden Draw() methods of the derived classes execute when the Draw()method is called using the DrawingObject base class reference from the dObj array. Here's what the output looks like:

Output:

I'm a Line.

I'm a Circle.

I'm a Square.

I'm just a generic drawing object.

The override Draw() method of each derived class executes as shown in the DrawDemo program. The last line is from the virtual Draw()method of the DrawingObject class. This is because the actual run-time type of the fourth array element was a DrawingObject object.

The code in this lesson can be compiled with the following command line:

csc DrawDemo.cs DrawingObject.cs Circle.cs Line.cs Square.cs

It will create the file DrawDemo.exe, which defaulted to the name of the first file on the command line.

### Summary

You should now have a basic understanding of polymorphism. You know how to define a virtual method. You can implement a derived class method that overrides a virtual method. This relationship between virtual methods and the derived class methods that overridethem enables polymorphism. This lesson showed how to use this relationship between classes to implement polymorphism in a program.