DELEGATES

A *delegate* is a C# language element that allows you to reference a method. If you were a C or C++ programmer, this would sound familiar because a *delegate* is basically a function pointer. However, developers who have used other languages are probably wondering, "Why do I need a reference to a method?". The answer boils down to giving you maximum flexibility to implement any functionality you want at runtime.

The following example shown how to use the delegate.

```
using System;

// this is the delegate declaration
public delegate int Comparer(object obj1, object obj2);

public class Name
{
   public string FirstName = null;
   public string LastName = null;

   public Name(string first, string last)
   {
      FirstName = first;
      LastName = last;
   }

   // this is the delegate method handler
   public static int CompareFirstNames(object name1, object name2)
   {
      string n1 = ((Name)name1).FirstName;
      string n2 = ((Name)name2).FirstName;

      if (String.Compare(n1, n2) > 0)
      {
         return 1;
      }
      else if (String.Compare(n1, n2) < 0)
      {
         return -1;
      }
      else
      {
         return 0;
      }
   }
```

```
    public override string ToString()
    {
        return FirstName + " " + LastName;
    }
}

class SimpleDelegate
{
    Name[] names = new Name[5];

    public SimpleDelegate()
    {
        names[0] = new Name("Joe", "Mayo");
        names[1] = new Name("John", "Hancock");
        names[2] = new Name("Jane", "Doe");
        names[3] = new Name("John", "Doe");
        names[4] = new Name("Jack", "Smith");
    }

    static void Main(string[] args)
    {
        SimpleDelegate sd = new SimpleDelegate();

        // this is the delegate instantiation
        Comparer cmp = new Comparer(Name.CompareFirstNames);

        Console.WriteLine("\nBefore Sort: \n");

        sd.PrintNames();

        // observe the delegate argument
        sd.Sort(cmp);

        Console.WriteLine("\nAfter Sort: \n");

        sd.PrintNames();
    }

    // observe  the delegate parameter
    public void Sort(Comparer compare)
    {
        object temp;

        for (int i=0; i < names.Length; i++)
        {
            for (int j=i; j < names.Length; j++)
```

```
            {
                // using delegate "compare" just like
                // a normal method
                if ( compare(names[i], names[j]) > 0 )
                {
                    temp = names[i];
                    names[i] = names[j];
                    names[j] = (Name)temp;
                }
            }
        }
    }

    public void PrintNames()
    {
        Console.WriteLine("Names: \n");

        foreach (Name name in names)
        {
            Console.WriteLine(name.ToString());
        }
    }
}
```

The first thing the program in Listing 14-1 does is declare a *delegate*. *Delegate* declarations look somewhat like methods, except they have the *delegate* modifier, are terminated with a semi-colon (*;*), and have no implementation. Below, is the *delegate* declaration from Listing 14-1.

```
public delegate int Comparer(object obj1, object obj2);
```

This *delegate* declaration defines the signature of a delegate handler method that this *delegate* can refer to. The delegate handler method, for the *Comparer delegate*, can have any name, but must have a first parameter of type *object*, a second parameter of type *object*, and return an *int* type. The following method from Listing 14-1 shows a delegate handler method that conforms to the signature of the *Comparer delegate*.

```
    public static int CompareFirstNames(object name1, object name2)
    {
        ...
    }
```