[0920] Non-Generic C# Collection Classes

When .NET was first released, it came with a set of Collection classes that C# made available via the System.Collections namespace.

System.Collection classes still exist in C# and you will find them used extensively in older code. Unfortunately, they have a limitation that has rendered them obsolete now - specifically they are not "type-safe" because they do not use a newer feature of C# called "Generics."

We'll talk about Generics and the newer, type-safe Collection classes in just a minute. For now, because System.Collection is still used - and because it illustrates the problem that Generics solves - let's look at it in more detail:

The System.Collections namespace contains the following classes and associated interfaces:

- **ArrayList** - ICollection, IEnumerable, IList
- **BitArray** - ICollection, IEnumerable
- **Hashtable** - ICollection, IEnumerable IDictionary
- **Queue** - ICollection, IEnumerable
- **SortedList** - ICollection, IEnumerable, IDictionary
- **Stack** - ICollection, IEnumerable

First off, notice that they all implement the ICollection interface - that makes sense because they are all part of the Collection family of classes. It also means that if we are writing a method that does something with "a collection" - any kind of collection - we can use "ICollection" as the data type in that method. Here's an example:

```
using System;
using System.Collections;

namespace LWTech.Testing
{
    public class CollectionArrayDemo
    {
        public static void Main()
        {
            ICollection[] myJunk = new ICollection[3];

            myJunk[0] = new ArrayList() { 3, 1, "4", 1, 5 };
            myJunk[1] = new Stack();
            ((Stack)myJunk[1]).Push("Bottom");
            ((Stack)myJunk[1]).Push("Middle");
            ((Stack)myJunk[1]).Push("Top");
            myJunk[2] = new Queue();
            ((Queue)myJunk[2]).Enqueue("First");
            ((Queue)myJunk[2]).Enqueue("Second");
            ((Queue)myJunk[2]).Enqueue("Third");
```

```
            foreach (ICollection c in myJunk)
            {
                string s = "";
                foreach (Object o in c)
                    s += o.ToString() + ",";
                Console.WriteLine(c.ToString() + ": " + s);
            }
        }
    }
}
```

Notice that in that program, I can stick any kind of C# collection into the "myJunk" array. The code doesn't care. Also notice that, in order to use one of the collections in the "myJunk" array as a real collection, I first have to cast it back to the appropriate collection's type. The output from this program is below:

```
System.Collections.ArrayList: 3,1,4,1,5,
System.Collections.Stack: Top,Middle,Bottom,
System.Collections.Queue: First,Second,Third,
```

OK, back to the System.Collection table above. Notice that all of these collections implement the IEnumerable interface. That means that you can use "foreach/in" on any of them - which I also demonstrate in the program above.

(Technically, you could use "IEnumerate" as the type for the "myJunk" array in the program (since they all implement it) however that would be misleading in this case since I want myJunk to be able to contain any "Collection", not just the one that can be enumerated.)

Now notice that only ArrayList implements the IList interface. That means that methods like Add(), Remove() and Get() are only available for ArrayList objects. Notice that, since they don't implement IList, I cannot use object initialization syntax on my Stack() and Queue() objects above because that requires the Add() method.

Similarly, Hashtable and SortedList implement the IDictionary interface meaning that they work with Key/Value pairs of data instead of single data items.

The Problem with System.Collections Classes

Look back at the program above and see if you can spot the problem with these classes. Here's a hint - What kind of data do I store in these collection objects?

In the program above, I am storing either ints or string objects inside my various collections. Take a closer look at the initializer for myJunk[0] - it takes BOTH ints and strings (i.e. "4"), right?

While at first this seems cool - "I can store anything in anything!" - it can be a problem if you want to make sure that you only store one type of data in a collection. What if I need to make sure that strings (or other objects) don't accidentally end up in my ArrayList object? There's no way to prevent that because there's no way for me to tell C# that I only want strings in my ArrayList.

Until...