



Unlocking the Power of Generics

Mark Strawmyer
Microsoft MVP



Agenda

- Traditional Collections and Drawbacks
- Value of Generics
- Syntax and Samples



Traditional Collections

- System.Collection namespace introduced in 1.0 version of .NET Framework
- Useful container types
 - Lists
 - Dictionaries
 - Hashtables
 - CollectionBase



Sample Class Definition

```
class TestItem
{
    private int itemValue = 0;
    public int ItemValue
    {
        get { return this.itemValue; }
        set { this.itemValue = value; }
    }

    public TestItem( int itemValue )
    {
        this.itemValue = itemValue;
    }
}
```



Sample ArrayList Use

```
ArrayList testCol = new ArrayList();
```

```
for( int i = 0; i < 20; i++)  
{  
    testCol.Add( new TestItem(i) );  
}
```

```
IEnumerator listEnum = testCol.GetEnumerator();  
while( listEnum.MoveNext() )  
{  
    Console.WriteLine(“{0}”,  
        ((TestItem)(listEnum.Current)).ItemValue);  
}
```



Drawbacks

- No type checking enforcement at compile time
 - Doesn't prevent adding unwanted types
 - Can lead to difficult to troubleshoot issues
- All items are stored as objects
 - Must be cast going in and coming out
 - Performance overhead of boxing and unboxing specific types

Enforcing Consistent Types

- Can control and eliminate unwanted types
- Create wrapper class for each type



Enforcing Consistent Types

```
private class TestItemCollection : CollectionBase
{
    public TestItem this[ int index ]
    {
        get { return (TestItem) List[index] ;}
        set { List[index] = value; }
    }

    public int Add( TestItem itemValue )
    {
        return List.Add(itemValue);
    }

    public void Remove( TestItem.itemValue )
    {
        List.Remove( itemValue );
    }
}
```





Enforcing Consistent Types

```
TestItemCollection testCol = new TestItemCollection ();
```

```
for( int i = 0; i < 20; i++)  
{  
    testCol.Add( new TestItem(i) );  
}
```

```
IEnumerator listEnum = testCol.GetEnumerator();  
while( listEnum.MoveNext() )  
{  
    Console.WriteLine(“{0}”,  
        ((TestItem)(listEnum.Current)).ItemValue);  
}
```



Drawbacks

- Previous example demonstrated enforcing consistent type use with 1.x
- Code generators such as CodeSmith can generate wrappers for you
- Larger code base even if it is by a code generator
- Still has performance overhead of boxing and unboxing



The Generic Solution

- Open constructed types
 - Class defined without a specific type
 - Type is specified when instantiated
- System.Collections.Generic namespace
 - List – array dynamically sized as needed
 - Linked list – doubly linked list
 - Queue – first in, first out collection of objects
 - Stack – last in, first out collection of objects



VB.NET Generics Sample

```
Dim testCol As New List(Of TestItem)()
```

```
For i as int32 = 0 To 19
```

```
    testCol.Add(New TestItem(i))
```

```
Next
```

```
For each item As TestItem in testCol
```

```
    Console.WriteLine("{0}", item.ItemValue)
```

```
Next
```




C# Generics Sample

```
List<TestItem> testCol = new  
    List<TestItem>();
```

```
for (int i = 0; i < 20; i++)  
{  
    testCol.Add( new TestItem(i) );  
}
```

```
foreach (TestItem item in testCol)  
{  
    Console.WriteLine(“{0}”, item.ItemValue);  
}
```

Socket Connection Pool C# Example



```
public static class SocketConnectionPool
{
    private static Queue<Socket> availableSockets = new Queue<Socket>();

    public static Socket GetSocket()
    {
        if (SocketConnectionPool.availableSockets.Count > 0)
        {
            Socket socket = null;
            while( SocketConnectionPool.availableSockets.Count > 0 )
            {
                socket = SocketConnectionPool.availableSockets.Dequeue();
                if (socket.Connected)
                    return socket;
            }
        }
        return SocketConnectionPool.OpenSocket();
    }

    public static void PutSocket(Socket socket) { ... }
    private static Socket OpenSocket() { ... }
}
```



Create Your Own Generics

- Generics can be applied a number of ways
 - Classes
 - `Public Class MyType(Of T, U)`
 - `public class MyType<T,U> {`
 - Method return types and parameters
 - `Public Function Foo() As T`
 - `public T Foo()`
 - Fields and properties
 - Indexers, properties, events, structs, interfaces, etc.

Generic Methods

- Can overload generic methods
 - Method signatures can be a challenge
 - Type parameters influence signature
 - Return type does not influence signature
 - Examples

```
Public List<I> Foo<I, J>(I val1, List<J> val2) { ... }
```

```
Public void Foo<K, L>(K val1, List<L> val2) { ... }
```

Constraints

- Can apply constraints to limit the types used for the type parameter

```
Public Class MyClass(Of T As {IMyClassA, IMyClassB})
```

```
...
```

```
End Class
```

```
Public class MyClass<T> where T : IMyClassA, IMyClassB
```

```
{
```

```
...
```

```
}
```



Default Values

- Assign a default value when the type is not known until implementation
 - Assigns null for reference types
 - Assigns appropriate default value for built-in types
 - Only available in C# for now

```
public class MyClass <K, V>
{
    public V Lookup(K key)
    {
        V retVal = default(V);
        ...
        return retVal;
    }
    ...
}
```



Nullable Types

- Built-in data types such as `int` don't offer a good way to check if it has been assigned a value
- `System.Nullable<T>` offers a nullable type

```
Nullable<int> intVal = null;
if (!intVal.HasValue)
{
    Console.WriteLine("Is null");
}
intVal = 0;
if (intVal.HasValue)
{
    Console.WriteLine("Has value");
}
```



Power Collections

- Open source library of generics
<http://www.wintellect.com/powercollections/>
- Microsoft support and participation
- Additional useful types
 - BigList
 - Deque
 - OrderedDictionary
 - OrderedSet
 - And more...



More Information

- “An Introduction to C# Generics” - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/csharp_generics.asp
- “Generics in .NET: Type Safety, Performance and Generality” - <http://www.codeguru.com/columns/dotnet/article.php/c9661/>
- Power Collections - <http://www.wintellect.com/powercollections/>