CS 683 Emerging Technologies

Spring Semester, 2003

Doc 25 C# Arrays, Indexers & Exceptions

Contents

References

Programming C#, Jesse Liberty, O'Reilly, Chapters 9, 11

C# Language Specification, http://download.microsoft.com/download/0/a/c/0acb3585-3f3f-4169-ad61-efc9f0176788/CSharp.zip

C# Class Library Detailed Specifications, http://download.microsoft.com/download/3/c/2/3c26966b-ce53-4d78-9076-5cb8331844f1/TypeLibrary.zip

A number of examples are taken directly from C# Language Specification and C# Class Library Detailed Specifications

Also see http://msdn.microsoft.com/net/ecma/ for more documentation

Arrays

System.Array is base class of all array types

Indexing starts at 0

Sample declarations

```
int[] anArray;
int[] semiInitilaized = new int[10];
int[] initilized = new int[3] {2,4,3};
int[] a = {2, 4, 3};

int[,] twoD = new int[4,5];
int[,] b = { {0, 1}, {2, 3}, {4, 5}};
b[0,1] = 19;

int[][] jagged = new int[4][];
jagged[0] = new int[10];
jagged[1] = new int[2];
jagged[2] = new int[20];
jagged[3] = new int[1];
```

Default Values

```
int[] anArray;
```

anArray is a null reference

Arrays of value types default to the default value for the type

```
int[] semiInitilaized = new int[10];
```

An array of size ten with all elements 0

Arrays of reference types default to null

```
Button[] buttons = new Button[5];
```

An array of size 5 with all elements null

Initializers are done at Compile time

```
int k = 3;
const int p = 3;
int[] a = new int[k];
int[] b = new int[p];
int[] c = new int[p] { 0, 1, 2 };
int[] d = new int[k] {0, 1, 2};              //Compile Error
```

foreach

Array iterator

```
public class Tester
	{
	public static void Main()
		{
		int[] a = new int[5];

		for (int k = 0; k < a.Length;k++)
			a[k] = k;

		foreach (int k in a )
			Console.WriteLine(k.ToString());

		}
	}
```

Aystem.Array
Properties

bool IList.IsFixedSize { get; }

bool IList.IsReadOnly { get; }

bool ICollection.IsSynchronized { get; }

public int Length { get; }

Gets the total number of elements in all the dimensions of the current instance.

public long LongLength {get;}

Gets the total number of elements in all the dimensions of the current instance

public int Rank { get; }

Gets the rank (number of dimensions) of the current instance.

object ICollection.SyncRoot { get; }

int ICollection.Count { get; }

Operations
BinarySearch

public static int BinarySearch(Array array, int index, int length, object value, IComparer comparer)

public static int BinarySearch(Array array, object value, IComparer comparer)

public static int BinarySearch(Array array, int index, int length, object value)

public static int BinarySearch(Array array, object value)

Binary Search Example

```
using System;
class BinarySearchExample {
  public static void Main() {
    int[] intAry = { 0, 2, 4, 6, 8 };
    Console.WriteLine( "The indices and elements of the array are: ");
    for ( int i = 0; i < intAry.Length; i++ )
      Console.Write("[{0}]: {1, -5}", i, intAry[i]);
    Console.WriteLine();
    SearchFor( intAry, 3 );
    SearchFor( intAry, 6 );
    SearchFor( intAry, 9 );
  }
  public static void SearchFor( Array ar, Object value ) {
    int i = Array.BinarySearch( ar, 0, ar.Length, value, null );
    Console.WriteLine();
    if ( i > 0 ) {
      Console.Write( "The object searched for, {0}, was found ", value );
      Console.WriteLine( "at index {1}.", value, i );
    }
    else if ( ~i == ar.Length ) {
      Console.Write( "The object searched for, {0}, was ", value );
      Console.Write( "not found,\nand no object in the array had " );
      Console.WriteLine( "greater value. " );
    }
    else {
      Console.Write( "The object searched for, {0}, was ", value );
      Console.Write( "not found.\nThe next larger object is at " );
      Console.WriteLine( "index {0}.", ~i );
    }
  }
}
```

Clone

public virtual object Clone()

Returns a **System.Object** that is a copy of the current instance.

Copy

public static void Copy(Array sourceArray, Array destinationArray, int length)

public static void Copy(Array sourceArray, int sourceIndex, Array destinationArray, int destinationIndex, int length

```
using System;
public class ArrayCopyExample {
  public static void Main() {
    int[] intAryOrig = new int[3];
    double[] dAryCopy = new double[3];
    for ( int i = 0; i < intAryOrig.Length; i++ )
      intAryOrig[i] = i+3;
    //copy the first 2 elements of the source into the destination
    Array.Copy( intAryOrig, dAryCopy, 2);
    Console.Write( "The elements of the first array are: " );
    for ( int i = 0; i < intAryOrig.Length; i++ )
      Console.Write( "{0,3}", intAryOrig[i] );
    Console.WriteLine();
    Console.Write( "The elements of the copied array are: " );
    for ( int i = 0; i < dAryCopy.Length; i++ )
      Console.Write( "{0,3}", dAryCopy[i] );
  }
}
```

CopyTo

```
public virtual void CopyTo(Array array, int index)
```

Copies all the elements of the current instance to the specified one-dimensional array starting at the specified subscript in the destination array.

GetEnumerator

```
public virtual IEnumerator GetEnumerator()
```

Returns a **System.Collections.IEnumerator** for the current instance.

Example

```
using System;
using System.Collections;
public class ArrayGetEnumerator {
  public static void Main() {
    string[,] strAry = {{"1","one"}, {"2", "two"}, {"3", "three"}};
    Console.Write( "The elements of the array are: " );
    IEnumerator sEnum = strAry.GetEnumerator();
    while ( sEnum.MoveNext() )
      Console.Write( " {0}", sEnum.Current );
  }
}
```

Output

```
The elements of the array are: 1 one 2 two 3 three
```

GetLowerBound

public int GetLowerBound(int dimension)

Returns the lower bound of the specified dimension in the current instance.

GetUpperBound

public int GetUpperBound(int dimension)

Returns the upper bound of the specified dimension in the current instance.

IndexOf

```
public static int IndexOf(Array array, object value, int startIndex, int
count)
public static int IndexOf(Array array, object value, int startIndex)
public static int IndexOf(Array array, object value)
```

Searches the specified one-dimensional **System.Array**, returning the index of the first occurrence of the specified **System.Object** in the specified range.

```
using System;
public class ArrayIndexOfExample {
   public static void Main() {
      int[] intAry = { 0, 1, 2, 0, 1 };
      Console.Write( "The values of the array are: " );
      foreach( int i in intAry )
         Console.Write( "{0,5}", i );
      Console.WriteLine();
      int j = Array.IndexOf( intAry, 1 );
      Console.WriteLine( "The first occurrence of 1 is at index
{0}", j );
   }
}
```

Output

```
The values of the array are: 0 1 2 0 1

The first occurrence of 1 is at index 1
```

LastIndexOf

public static int LastIndexOf(Array array, object value, int startIndex, int count)

public static int LastIndexOf(Array array, object value, int startIndex)

public static int LastIndexOf(Array array, object value)

Example

```
using System;

public class ArrayLastIndexOfExample {

   public static void Main() {
      int[] intAry = { 0, 1, 2, 0, 1 };
      Console.Write( "The values of the array are: ");
      foreach( int i in intAry )
         Console.Write( "{0,5}", i );
      Console.WriteLine();
      int j = Array.LastIndexOf( intAry, 1 );
      Console.WriteLine( "The last occurrence of 1 is at index {0}", j );
   }
}
```

Output

```
The values of the array are: 0 1 2 0 1


The last occurrence of 1 is at index 4
```

Reverse

public static void Reverse(Array array, int index, int length)
public static void Reverse(Array array)

```
using System;
public class ArrayReverseExample {
   public static void Main() {
      string[] strAry = { "one", "two", "three" };
      Console.Write( "The elements of the array are:");
      foreach( string str in strAry )
         Console.Write( " {0}", str );
      Array.Reverse( strAry );
      Console.WriteLine();
      Console.WriteLine( "After reversing the array," );
      Console.Write( "the elements of the array are:");
      foreach( string str in strAry )
         Console.Write( " {0}", str );
   }
}
```

Output

```
The elements of the array are: one two three


After reversing the array,


the elements of the array are: three two one
```

Sort

public static void Sort(Array keys, Array items, int index, int length, IComparer comparer)
public static void Sort(Array array, int index, int length, IComparer comparer)
public static void Sort(Array keys, Array items, IComparer comparer)
public static void Sort(Array array, IComparer comparer)
public static void Sort(Array keys, Array items, int index, int length)
public static void Sort(Array array)
public static void Sort(Array keys, Array items)
public static void Sort(Array array, int index, int length)

Example

```
using System;
public class ArraySortExample {
   public static void Main() {
      string[] strAry = { "All's", "well", "that", "ends", "well" };
      Console.Write( "The original string array is: " );
      foreach ( String str in strAry )
         Console.Write( str + " " );
      Console.WriteLine();
      Array.Sort( strAry );
      Console.Write( "The sorted string array is: " );
      foreach ( string str in strAry )
         Console.Write( str + " " );
   }
}
```

Output

```
The original string array is: All's well that ends well
The sorted string array is: All's ends that well well
```

CreateInstance

```
public static Array CreateInstance(Type elementType, int[] lengths)
public static Array CreateInstance(Type elementType, int length1, int
length2, int length3)
public static Array CreateInstance(Type elementType, int length1, int
length2)
public static Array CreateInstance(Type elementType, int length)
public static Array CreateInstance(Type elementType, int[] lengths,
int[] lowerBounds)
```

Creates a zero-based, multidimensional array of the specified **System.Type** and dimension lengths.

GetValues

public object GetValue(int[] indices)
public object GetValue(int index1, int index2)
public object GetValue(int index1, int index2, int index3)

Gets the value at the specified position in the current multidimensional instance.

public object GetValue(int index)

Gets the value at the specified position in the current one-dimensional instance.

```
using System;
public class ArrayGetValueExample {
   public static void Main() {
      String[] strAry = { "one", "two", "three", "four", "five" };
      Console.Write( "The elements of the array are: " );
      for( int i = 0; i < strAry.Length; i++ )
         Console.Write( " '{0}' ", strAry.GetValue( i ) );
   }
}
```

Output

```
The elements of the array are: 'one' 'two' 'three' 'four' 'five'
```

Initialize

```
public void Initialize()
```

Initializes every element of the current instance of value-type objects by calling the default constructor of that value type.

SetValue

```
public void SetValue(object value, int index)
public void SetValue(object value, int index1, int index2)
public void SetValue(object value, int index1, int index2, int index3)
public void SetValue(object value, int[] indices)
```

Indexers

Allows objects to be indexed like an array

Similar to C++ overloading of [ ] operator

```
public class Foo
	{
	int[] data = {0, 1, 2 ,3, 4};
	public int this[int index]
		{
		get
			{
			return data[index];
			}

		set
			{
			data[index] = value;
			}
		}
	}

public class Tester
	{
	public static void Main()
		{
		Foo test = new Foo();
		test[2] = 12;
		int result = test[1];
		}
	}
```

Example

```
using System;
class BitArray
{
   int[] bits;
   int length;
   public BitArray(int length) {
      if (length < 0) throw new ArgumentException();
      bits = new int[((length - 1) >> 5) + 1];
      this.length = length;
   }
   public int Length {
      get { return length; }
   }
   public bool this[int index] {
      get {
         if (index < 0 || index >= length) {
            throw new IndexOutOfRangeException();
         }
         return (bits[index >> 5] & 1 << index) != 0;
      }
      set {
         if (index < 0 || index >= length) {
            throw new IndexOutOfRangeException();
         }
         if (value) {
            bits[index >> 5] |= 1 << index;
         }
         else {
            bits[index >> 5] &= ~(1 << index);
         }
      }
   }
}
```

Using the Indexer

```
class CountPrimes
{
   static int Count(int max) {
      BitArray flags = new BitArray(max + 1);
      int count = 1;
      for (int i = 2; i <= max; i++) {
         if (!flags[i]) {
            for (int j = i * 2; j <= max; j += i) flags[j] = true;
            count++;
         }
      }
      return count;
   }
   static void Main(string[] args) {
      int max = int.Parse(args[0]);
      int count = Count(max);
      Console.WriteLine("Found {0} primes between 1 and {1}", count,
max);
   }
}
```

Index arguments can be any type

```
using System;
class Grid
{
   const int NumRows = 26;
   const int NumCols = 10;
   int[,] cells = new int[NumRows, NumCols];

   public int this[char c, int colm]
   {
      get {
         c = Char.ToUpper(c);
         if (c < 'A' || c > 'Z') {
            throw new ArgumentException();
         }
         if (colm < 0 || colm >= NumCols) {
            throw new IndexOutOfRangeException();
         }
         return cells[c - 'A', colm];
      }
      set {
         c = Char.ToUpper(c);
         if (c < 'A' || c > 'Z') {
            throw new ArgumentException();
         }
         if (colm < 0 || colm >= NumCols) {
            throw new IndexOutOfRangeException();
         }
         cells[c - 'A', colm] = value;
      }
   }
}
```

Overloading Indexers

```
public class Foo
	{
	int[] data = {0, 1, 2 ,3, 4};
	public int this[int index]
		{
		get
			{
			return data[index];
			}

		set
			{
			data[index] = value;
			}
		}

	public int this[string index]
		{
		get
			{
			switch (index)
				{
				case "one": return this[1];
				case "two": return this[2];
				case "three" : return this[3];
				}
			return 10;
			}
		}
	}
```

Exceptions

Similar to Java Exceptions

```
public class Tester
	{
	public static void Main()
		{
		try
			{
			int y = 0;
			int x = 12/y;
			}
		catch (System.DivideByZeroException e)
			{
			Console.WriteLine( e.Message);
			}
		catch (System.ArithmeticException)
			{
			Console.WriteLine( "Math error");
			}
		catch
			{
			Console.WriteLine("All other errors");
			}
		finally
			{
			Console.WriteLine("Always done");
			}
		}
	}
```

Some Difference With Java

Catch does not need argument

General purpose catch with no type or argument

Does not have checked and uncheck exceptions

Throwing an Exception

```
public class Tester
    {
    public static void Main()
        {
        throw new System.DivideByZeroException( “A test”);
        }
    }
```

Exception Properties

public virtual string Message { get; }

Returns a string that describes the current Exception.

public virtual string StackTrace { get; }

Returns a string representation of the frames on the call stack at the time the current Exception was thrown.

public Exception InnerException { get; }

When an exception is rethrown, InnerException returns the original exception

Rethrowing an Exception

```
public class Tester
    {
    public static void Main()
        {
        try
            {
            int y = 0;
            int x = 12/y;
            }
        catch (System.DivideByZeroException )
            {
            throw;
            }
        }
    }
```

New Exception Type

```
using System;

public class EndOfSemesterException : System.ApplicationException
	{
	public EndOfSemesterException() : base(){}

	public EndOfSemesterException(string message) : base(message) {}

	}
```