2025-08-27

# List as ICollection

Another way of implementing lists is to make our class realize the [ICollection interface](https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.icollection-1?view=net-9.0):



A UML diagram for the ICollectionT class ([text version](https:///princomp.github.io/diag/cla/ICollection.txt), [image version](https:///princomp.github.io/diag/cla/ICollection.png), [svg version](https:///princomp.github.io/diag/cla/ICollection.svg))

This requires implementing a series of properties and methods:

 // Empty
 public bool IsEmpty()
 {
 return first == null;
 }

 // Add is simply "AddF", slightly revisited.
 public void Add(T value)
 {
 if (isReadonly)
 {
 throw new InvalidOperationException(
 "List is read-only."
 );
 }
 Cell cCell = first;
 if (cCell != null)
 {
 while (cCell.Next != null)
 // As long as the cCell Cell has a neighbour…
 {
 cCell = cCell.Next;
 // We move the cCell cell to this neighbour.
 }
 cCell.Next = new Cell(value, null);
 }
 else
 {
 first = new Cell(value, null);
 }
 }

 public void Clear()
 {
 first = null;
 }

 public bool Contains(T value)
 {
 bool found = false;
 Cell cCell = first;
 while (cCell != null && !found)
 {
 if (cCell.Data.Equals(value))
 {
 found = true;
 }
 cCell = cCell.Next;
 }
 return found;
 }

 // Copies the elements of the ICollection to an Array, starting at a particular Array index.
 public void CopyTo(T[] array, int arrayIndex)
 {
 if (array == null)
 throw new ArgumentNullException(
 "The array cannot be null."
 );
 if (arrayIndex < 0)
 throw new ArgumentOutOfRangeException(
 "The starting array index cannot be negative."
 );
 if (Count > array.Length - arrayIndex)
 throw new ArgumentException(
 "The destination array has fewer elements than the collection."
 );

 Cell cCell = first;
 int i = 0; // keeping track of how many elements were copied.
 while (cCell != null)
 {
 array[i + arrayIndex] = cCell.Data;
 i++;
 cCell = cCell.Next;
 }
 }

 public bool Remove(T value)
 {
 if (isReadonly)
 {
 throw new InvalidOperationException(
 "List is read-only"
 );
 }
 bool removed = false;
 if (!IsEmpty())
 {
 if (first.Data.Equals(value))
 {
 first = first.Next;
 removed = true;
 }
 else
 {
 Cell cCell = first;
 while (cCell.Next != null)
 {
 if (cCell.Next.Data.Equals(value))
 {
 cCell.Next = cCell.Next.Next;
 removed = true;
 }
 }
 }
 }
 return removed;
 }

 public int Count
 {
 get
 {
 int size = 0;
 Cell cCell = first;
 while (cCell != null)
 {
 cCell = cCell.Next;
 size++;
 }
 return size;
 }
 }

 public bool isReadonly = false;
 public bool IsReadOnly
 {
 get { return isReadonly; }
 set { isReadonly = value; }
 }

 public IEnumerator<T> GetEnumerator()
 {
 Cell cCell = first;
 while (cCell != null)
 {
 yield return cCell.Data;
 cCell = cCell.Next;
 }
 }

 IEnumerator IEnumerable.GetEnumerator()
 {
 return this.GetEnumerator(); // call the generic version of the method
 }

[*(Download this code)*](https:///princomp.github.io/code/projects/CList_ICollection.zip)