C# – 4

by

Dr. Günter Kolousek

Interface IFormattable

```
using System;
using System.Collections.Generic;
class Person : IFormattable {
  public string first_name; public string last_name;
  public Person(string _first_name, string _last_name) {
      first_name = _first_name; last_name = _last_name;  }
  public override string ToString()
    => $"{first_name} {last_name}";
  public string ToString(string format,
                         IFormatProvider fmt_provider) {
    if (format == null) format = "N";
    switch (format.ToUpper()) {
      case "N": return ToString();
      case "F": return first_name;
      case "L": return last_name;
      default:
        throw new FormatException(String.Format(fmt_provider,
            $"Format {format} is not supported"));
    } }
  public string ToString(string format)
    => ToString(format, null);  }
```

Interface IFormattable - 2

```
public class Program {
    public static void Main() {
        Person p;
        p.first_name = ”otto”;
        p.last_name = ”meier”;
        System.Console.WriteLine(p);
        System.Console.WriteLine($”{p:F}”);
        System.Console.WriteLine($”{p:L}”);
    }
}
```

Interface IComparable

```
class Person : IFormattable, IComparable<Person> {
    // ...
    public int CompareTo(Person o) {
        int cmp=last_name?.CompareTo(o?.last_name) ?? -1;
        if (cmp == 0)
            return first_name?.CompareTo(o?.first_name) ?? -1;
        return cmp;
    }
}
public static void Main() {
    List<Person> lst=new List<Person>{
        new Person(”otto”, ”meier”),
        new Person(”franz”, ”meier”),
        new Person(”sam”, ”muster”)
    };
    lst.Sort();
    foreach (var o in lst)
        Console.WriteLine($”{o}”);
} }
```

Sortieren mittels Comparison<T>

```
using System;  using System.Collections.Generic;
class Person {
  public string first_name;
  public string last_name;
  public Person(string _first_name, string _last_name) {
      first_name = _first_name;
      last_name = _last_name;   }
  public override string ToString() =>
    $"{first_name} {last_name}";  }
public class Program {
  public static void Main() {
    Person p=new Person("otto", "meier");
    List<Person> lst=new List<Person>{
        new Person("sam", "muster"),
        new Person("otto", "meier"),
        new Person("franz", "meier")
    };
    // *in-place* mittels    Comparison<Person> (delegate)
    lst.Sort((x, y) => x.last_name.CompareTo(y.last_name));
    foreach (var o in lst) Console.WriteLine(o); } }
```

Sortieren mittels LINQ

```
using System; using System.Collections.Generic;
using System.Linq;
class Person {
  public string first_name;
  public string last_name;
  public Person(string _first_name, string _last_name) {
      first_name = _first_name;
      last_name = _last_name;  }
  public override string ToString() =>
    $"{first_name} {last_name}";  }
public class Program {
  public static void Main() {
    Person p=new Person("otto", "meier");
    List<Person> lst=new List<Person>{
        new Person("sam", "muster"),
        new Person("otto", "meier"),
        new Person("franz", "meier")
    };  // new List will be created out of an Enumerator v
    List<Person> sorted=lst.OrderBy(o=>o.last_name).ToList();
    foreach (var o in lst) Console.WriteLine(o);  } }
```

Überblick Collections

- Namensräume
  - `System.Collections.Generic`
    - (`System.Collections`)
- Interfaces
  - `ICollection<T>`
    - `IList<T>`
    - `ISet<T>`
    - `IDictionary<TKey,TValue>`
- Interfaces – 2
  - `IEnumerable<T>`
  - `IComparer<T>`
    - wenn mehrere Sortierkriterien benötigt werden
    - d.h. je Kriterium eine Klasse, die `IComparer<T>` implementiert
    - Methode: `int Compare(T, T)`
    - überladene Methode `Sort`!

Überblick Collections – 2

- Klassen
  - `List<T>`
  - `Queue<T>`
  - `Stack<T>`
  - `LinkedList<T>`
  - `Dictionary<T>`
  - `Set<T>`
  - `BitArray` bzw. `BitVector32`

`IEnumerable`, `IEnumerator`

- in `System.Collections.Generic`
- `IEnumerable<T>`
  - leitet ab von `System.Collections.Enumerable`
  - Methoden
    - `IEnumerator<T> GetEnumerator()`
- `IEnumerator<T>`
  - leitet ab von `System.Collections.IEnumerator` und `IDisposable`
  - Property: `T Current { get }`
  - Methoden
    - `bool MoveNext()` ... `false` → Ende!
    - `void Reset()` ... setzt an den Anfang
    - `void Dispose()` ... um Ressourcen freizugeben

IEnumerable, IEnumerator - 2

```
using System;
using static System.Console;
using System.Collections.Generic;

public class Program {
  public static void Main() {
    string[] lines={"abc", "def", "ghi"};
    foreach (var line in lines)
            WriteLine(line);
    IEnumerator<string> lines_enum=
      ((IEnumerable<string>)lines).GetEnumerator();
    //lines.GetEnumerator()->System.Collections.IEnumerator !
    while (lines_enum.MoveNext())
        WriteLine(lines_enum.Current);
  }
}
```

List

- Sequenz, Größe veränderbar, mittels Index zugreifbar
- zusätzliche wichtige Methoden:
  - `SequenceEqual(List)`
    - `Equals` vergleicht Referenzen!!!
  - `ReadOnlyCollection<T> AsReadOnly()`
  - `void Clear()`
    - auch für andere Collections
  - `bool Contains(T)`
    - auch für andere Collections
  - `T Find(Predicate<T>)`
  - `List<T> FindAll(Predicate<T>)`
  - `void ForEach(Action<T>)`
  - `List<T>.Enumerator GetEnumerator()`
    - auch für andere Collections
  - `int IndexOf(T)`
  - `void Reverse()`
  - `T[] ToArray()`
    - auch für andere Collections
  - `bool TrueForAll(Predicate<T>)`

List-2

```
using System;
using System.Collections.Generic;
public class Program {
    public static void Main() {
        List<int> even=new List<int>();
        even.Add(0);
        even.AddRange(new int[]{4,6,8,10});  // IEnumerable!
        even.Insert(1, 2);
        WriteLine(even[1]);  // -> 2
        even.Remove(0);
        foreach (var i in even)
            Console.Write($"{i} ");  // -> 2 4 6 8 10
        Console.WriteLine();
        List<int> odd=new List<int>{1,3,5,7,9,11};
        odd.RemoveAt(odd.FindIndex(i => i == 11));
        var all=new List<int>();
        all.AddRange(even);  all.AddRange(odd);
        all.Sort();
        foreach (var i in all)
            Console.Write($"{i} ");  // 1 2...9 10
    } }
```

List-3

```
// Person as before...
using System;
using static System.Console;
using System.Collections.Generic;

public class Program {
    public static void Main() {
        List<Person> persons=new List<Person>();
        persons.Add(new Person("mini", "meier"));
        persons.AddRange(new Person[]{
              new Person("maxi", "meier"),
              new Person("mini", "osterhase"),
              new Person("otto", "huber")
          });
        // will not remove anything: Equals not impl!
        persons.Remove(new Person("otto", "huber"));
        // will be sorted correctly -> CompareTo
        persons.Sort();
        foreach (var p in persons)
              WriteLine($"{p}");
    } }
```

Queue

```
using System;
using static System.Console;
using System.Collections.Generic;

public class Program {
    public static void Main() {
        Queue<int> q=new Queue<int>();
        q.Enqueue(0);
        q.Enqueue(1);
        q.Enqueue(2);
        q.Enqueue(3);
        foreach (var i in q)
            Write($"{i} ");
        WriteLine();
        WriteLine(q.Count);  // -> 4
        WriteLine(q.Peek());  // -> 0
        WriteLine(q.Count);  // -> 4
        WriteLine(q.Dequeue());  // -> 0
        WriteLine(q.Count);  // -> 3
```

Queue – 2

```
            // -> InvalidOperationException: Collection was
            // modified after the enumerator was instantiated
            //foreach (var i in q)
            //    q.Dequeue();

            //for (int i=0; i<q.Count; i++)
            //    q.Dequeue();
            //WriteLine(q.Count);  // -> 1

            int cnt=q.Count;  // do it this way!
            for (int i=0; i<cnt; i++)
                q.Dequeue();
            WriteLine(q.Count);  // -> 0
        }
    }
```

Stack

```
using System;
using static System.Console;
using System.Collections.Generic;
public class Program {
    public static void Main() {
        Stack<int> q=new Stack<int>();
        q.Push(0);
        q.Push(1);
        q.Push(2);
        q.Push(3);
        foreach (var i in q)
            Write($"{i} ");
        WriteLine();
        WriteLine(q.Count);  // -> 4
        WriteLine(q.Peek());  // -> 3
        WriteLine(q.Count);  // -> 4
        WriteLine(q.Pop());  // -> 3
        WriteLine(q.Count);  // -> 3
        q.Clear();
        WriteLine(q.Count);  // -> 0
    } }
```

LinkedList

- im Regelfall ist `List` zu verwenden...
- außer
  - hinzufügen/entfernen in der ”Mitte”
    - einfügen/entfernen am Anfang
  - umkehren der Reihenfolge
- Kein Indexer vorhanden,...
- Minibeispiel

```
LinkedList<int> ll=new LinkedList<int>();
ll.AddLast(3);
ll.AddLast(2);
ll.AddLast(1);
// LinkedListNode -> .Value
Console.WriteLine(ll.First.Value);
foreach (var i in ll)
    Console.WriteLine(i);
```

Dictionary

```
using System;
using static System.Console;
using System.Collections.Generic;

public class Program {
  public static void Main() {
    var phonebook=new Dictionary<string, int>(){
        ["maxi"]=4711,
        ["mini"]=4712  };
    WriteLine(phonebook["mini"]);
    try {
        WriteLine(phonebook["otto"]);
    } catch (KeyNotFoundException e) {
        WriteLine(e.Message);  }
    foreach (var key in phonebook.Keys) Write($"{key} ");
    foreach (var value in phonebook.Values) Write($"{value} ")
    phonebook["otto"] = 4713;
    phonebook.Remove("mini");
    WriteLine(phonebook.ContainsKey("mini")); // -> False
    WriteLine(phonebook.ContainsValue(4713)); // -> True
  } }
```

HashSet

```
using System;
using static System.Console;
using System.Collections.Generic;
public class Program {
    public static void Main() {
        var a=new HashSet<int>{1,2,3};
        var b=new HashSet<int>{2,3,4};
        if (!b.Add(4))
            WriteLine("4 already in set b!");
        WriteLine(a.IsSubsetOf(b)); // -> False
        WriteLine(b.IsSupersetOf(b)); // -> True
        WriteLine(a.Overlaps(b));  // -> True
        var a2=new HashSet<int>(a);
        a2.IntersectWith(b);
        foreach (var i in a2)
            Write($"{i} ");  // -> 2 3
        a2.UnionWith(b);  // a2 = {2,3,4}
```

HashSet - 2

```
    WriteLine(a2.Contains(3)); // -> True
    a2.Remove(3);
    a2.RemoveWhere(x => x % 2 == 0);
    WriteLine(a2.Count == 0);  // -> True
    a.ExceptWith(b);
    foreach (var i in a)
        Write($"{i} ");  // -> 1
    a.SymmetricExceptWith(b);
    foreach (var i in a)
        Write($"{i} ");  // -> 1 2 3 4
} }
```

ObservableCollection

```
using System;
using static System.Console;
using System.Collections.Specialized;
using System.Collections.ObjectModel;

public class Program {
  public static void data_changed(object sender,
                          NotifyCollectionChangedEventArgs e) {
    Write($"{e.Action.ToString()}: ");
    if (e.OldItems != null) {
      Write($"starting old index: {e.OldStartingIndex}, ");
      Write($"old items: ");
      foreach (var item in e.OldItems) WriteLine($"{item} ");
    }
    if (e.NewItems != null) {
      Write($"starting new index: {e.NewStartingIndex}, ");
      Write($"new items: ");
      foreach (var item in e.NewItems) WriteLine($"{item} ");
    }
    WriteLine();  }
```

ObservableCollection-2

```
  public static void Main() {
    var data=new ObservableCollection<string>();
    data.CollectionChanged += data_changed;
    data.Add("first");
    // -> Add: starting new index: 0, new items: first
    data.Add("second");
    // -> Add: starting new index: 1, new items: second
    data.Insert(1, "third");
    // -> Add: starting new index: 1, new items: third
    data.Remove("first");
    // -> Remove: starting old index: 0, old items: first
  }
}
```

Dateien und Verzeichnisse

- `DriveInfo`
- Utility-Klassen
  - `File` … wenn Produktivität
  - `Directory` … wenn Produktivität
  - `Path`
- wenn Performance:
  - `DirectoryInfo`
  - `FileInfo`

DriveInfo

```
using System;
using static System.Console;
using System.IO;

public class Program {
  public static void Main() {
    DriveInfo[] drives=DriveInfo.GetDrives();
    foreach (var drive in drives) {
      WriteLine(drive.Name);  // -> / or C:\
      WriteLine(drive.DriveFormat);  // -> ext or NTFS
      WriteLine(drive.DriveType);  // -> Fixed or Removable
      WriteLine(drive.RootDirectory);  // -> / or C:\
      WriteLine(drive.VolumeLabel);  // -> / or Windows
      WriteLine(drive.TotalFreeSpace);
      WriteLine(drive.AvailableFreeSpace);
      WriteLine(drive.TotalSize);
      WriteLine();
    }
  }
}
```

File

```
using System;
using static System.Console;
using System.IO;
public class Program {
  public static void Main() {
    File.Create(@"/tmp/test.txt");  // empty!
    File.Delete(@"/tmp/test.txt");
    WriteLine(File.Exists(@"/tmp/test.txt"));  // -> False
    File.WriteAllLines(@"/tmp/test.txt", new string[]{"a","b"}
    File.AppendAllText(@"/tmp/test.txt", "c\nd\n");
    string[] lines=File.ReadAllLines(@"/tmp/test.txt");
    string text=File.ReadAllText(@"/tmp/test.txt");
    WriteLine(text); // -> a\nb\nc\nd\n\n
    File.Delete(@"/tmp/test2.txt");
    File.Move(@"/tmp/test.txt", @"/tmp/test2.txt");
    File.Copy(@"/tmp/test2.txt", @"/tmp/test.txt");
    try {
        File.Copy(@"/tmp/test2.txt", @"/tmp/test.txt");
    } catch (IOException) {
        WriteLine("already exists"); }
    // true -> overwrite
    File.Copy(@"/tmp/test2.txt", @"/tmp/test.txt", true); } }
```

Directory

```
using System;  using static System.Console;
using System.IO;  using static System.IO.Directory;
public class Program {
  public static void Main() {
    CreateDirectory(”/tmp/data”);
    try { Delete(”/tmp/data”); } catch (IOException) {
        WriteLine(”not emty”); }
    // true -> recursive
    if (Exists(”/tmp/data”)) Delete(”/tmp/data”, true);
    CreateDirectory(”/tmp/data”);
    File.Create(@”/tmp/data/test.txt”);
    WriteLine(Exists(”/tmp/data/test.txt”));  // -> False
    if (Exists(”/tmp/dat”)) Delete(”/tmp/dat”, true);
    Move(”/tmp/data”, ”/tmp/dat”);
    foreach (var dir in EnumerateDirectories(”/tmp/dat”))
        WriteLine(dir);  // -> /tmp/dat/data2
    string[] dirs=GetDirectories(”/tmp/dat”);
    foreach (var file in EnumerateFiles(”/tmp/dat”))
        WriteLine(file);  // -> /tmp/dat/test.txt
    WriteLine(GetCurrentDirectory());  // z.B. /home/maxi
    WriteLine(GetParent(GetCurrentDirectory())); // z.B. /home
  } }
```

Path

```
using System;    using static System.Console;
using System.IO; using static System.IO.Path;
public class Program {
  public static void Main() {
    WriteLine(Combine(GetTempPath(), "test.txt"));
    string[] labels=new string[]{"/tmp", "data", "test.dat"};
    string p=Combine(labels);
    WriteLine(p);  // -> /tmp/data/test.dat
    p2 = ChangeExtension(p, "txt");
    WriteLine(GetExtension(p));  // -> .txt
    WriteLine(GetFileName(p));  // -> test.txt
    WriteLine(GetFileNameWithoutExtension(p));  // -> test
    WriteLine(GetDirectoryName(p));  // -> /tmp/data
    WriteLine(GetTempFileName());// z.B.: /tmp/tmp38c55ebc.tmp
    WriteLine(GetTempPath());  // -> /tmp/
    // kryptographisch sicher:
    WriteLine(GetRandomFileName());  // z.B.: vf1ernld.b7e
    WriteLine(HasExtension("/tmp/data"));  // -> False
    WriteLine(IsPathRooted("/tmp/data/test.txt"));  // -> True
    WriteLine(IsPathRooted("data/test.txt"));  // -> False
  } }
```

FileInfo

```
using System;
using static System.Console;
using System.IO;

public class Program {
    public static void Main() {
        var f=new FileInfo(”/tmp/test.txt”);
        f.CreateText();
        f.Refresh();  // !
        WriteLine(f.Name);  // -> test.txt
        WriteLine(f.Extension);  // -> .txt
        WriteLine(f.Directory); // DirectoryInfo! -> /tmp
        WriteLine(f.DirectoryName);  // -> /tmp
        WriteLine(f.FullName);  // -> /tmp/test.text
        WriteLine(f.IsReadOnly);  // -> False
        WriteLine(f.Length);  // -> 0
        WriteLine(f.CreationTime);  // -> 17.07.2018 09:10:55
        WriteLine(f.LastAccessTime);
        WriteLine(f.LastWriteTime);
    }
}
```

FileInfo - 2

```
using System;
using static System.Console;
using System.IO;

public class Program {
    public static void Main() {
        var f=new FileInfo(”/tmp/test.txt”);
        StreamWriter w=f.CreateText();  // UTF-8
        WriteLine(w.Encoding);  // -> System.Text.UTF-8Encodin
        w.Write(”abc”);
        w.WriteLine('d');
        w.Write(4711);
        // ... a la Console.Write
        w.Close();
        StreamReader r=f.OpenText();  // UTF-8
        Write((char)r.Read());  // next char -> a
        WriteLine(r.ReadLine());  // -> bcd
        WriteLine(r.ReadToEnd());  // -> 4711
    }
}
```

Streams

- Abstrakte Basisklasse `Stream` (byteorientiert!)
  - `FileStream`
  - `MemoryStream`
  - `NetworkStream`
  - `BufferedStream` ... Ref auf `Stream`
  - Basisklasse `PipeStream` ... zur Kommunikation zwischen Prozessen
    - → `NamedPipeServerStream` und `NamedPipeClientStream`
    - → `AnonymousPipeServerStream` und `AnonymousPipeClientStream`
  - `CryptoStream` ... Ref auf Stream
- Abstrakte Basisklasse `TextReader`
  - `StringReader`
  - `StreamReader` ... Ref auf `Stream`
- Abstrakte Basisklasse `TextWriter`
  - `StringWriter`
  - `StreamWriter` ... Ref auf `Stream`

Streams – 2

- zum Versenden von binären Werten (im Prinzip `byte`, `int`, `bool`, `float`, `double`, `char`)
  - `BinaryReader` … Ref auf `Stream`
  - Liest von Stream in einen angegebenen Kodierung
    - `ASCIIEncoding`, `UTF8Encoding`, `UnicodeEncoding` (repräsentiert UTF-16), `UTF32Encoding`)
  - `BinaryWriter` … Ref auf `Stream`
- zum Komprimieren
  - `DeflateStream`
  - `GZipStream` verwendet `DeflateStream`
  - `BrotliStream` … von Google
- Komprimieren und Archivieren
  - `ZipArchive`

Streams – 3

```
using System;
using static System.Console;
using System.IO;
public class Program {
    public static void Main() {
        var f=new FileInfo("/tmp/test.txt");

        // =using= ensures that Dispose() is automatically
        // called if object implements IDisposable
        using (TextWriter writer=f.CreateText()) {
            writer.WriteLine("abc");
            writer.WriteLine("def");
        }
        using (TextWriter writer=f.AppendText()) {
            writer.WriteLine("ghi");
        }
        using (TextReader reader=f.OpenText()) {
            while (reader.Peek() >= 0)
                WriteLine(reader.ReadLine());
        } } }
```

Streams – 4

```
using System;
using static System.Console;
using System.IO;

public class Program {
    public static void Main() {
        var f=new FileInfo("test.txt");

        using (TextReader reader=f.OpenText()) {
            string line;
            while ((line = reader.ReadLine()) != null)
                WriteLine(line);
        }
        foreach (var line in File.ReadLines("test.txt"))
                WriteLine(line);
        // no method ReadLines in TextReader!!!
    } }
```

Streams – 5

```
using System;  using static System.Console;
using System.IO;
public class Program {
  public static void Main() {
    try {
        using (FileStream f=new FileStream("/tmp/random.dat",
                                           FileMode.Create)) {
            for (byte i=65; i < 65+26; ++i)
                f.WriteByte(i);
            f.Seek(7, SeekOrigin.Begin);
            Write((char)f.ReadByte());
            f.Seek(-4, SeekOrigin.Current);
            Write((char)f.ReadByte());
            f.Seek(6, SeekOrigin.Current);
            Write((char)f.ReadByte());
            f.Seek(-1, SeekOrigin.Current);
            Write((char)f.ReadByte());
            f.Seek(2, SeekOrigin.Current);
            Write((char)f.ReadByte());
        }
    } catch (Exception e) { WriteLine(e.Message); } } }
```

IDisposable

```
using System;  using static System.Console;
using System.IO;
public class RessourceHolder : IDisposable {
  private bool disposed=false;
  public void Dispose() {
    Dispose(true);
    GC.SuppressFinalize(this); }
  protected virtual void Dispose(bool disposing) {
    if (!disposed) {
        // false -> called by runtime from inside finalizer
        if (disposing) {
            WriteLine(”cleanup managed objects”);
        }
        WriteLine(”unmanaged objects: set large fields to null
        disposed = true;
    } }
  ~RessourceHolder() {
      Dispose(false); } }
public class Program {
  public static void Main() {
      using (var rh=new RessourceHolder()) {} } }
```

Attribute

- siehe `enum`
- wie Annotations in Java
- Information an Klasse, Methode
- Attribute können mittels Reflection abgefragt werden
- vordefiniert bzw. benutzerdefiniert

Serialisierung

```
using System;
using static System.Console;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class Employee {
    public int id;
    public string name;
    [NonSerialized]
    public int salary;

    public Employee() {
        id = 0;
        name = null;
        salary = 0;
    }
}
```

Serialisierung – 2

```
public class Program {
  public static void Main() {
    //Create a new Employee object
    Employee o=new Employee();
    o.id = 1; o.name = ”Maxi”; o.salary = 4567;

    Stream stream=new FileStream(”employee.dat”,
      FileMode.Create, FileAccess.Write);
    BinaryFormatter bformatter = new BinaryFormatter();
    bformatter.Serialize(stream, o);
    stream.Close();

    o = null;
    stream = new FileStream(”employee.dat”,
      FileMode.Open, FileAccess.Read);
    bformatter = new BinaryFormatter();
    o = (Employee)bformatter.Deserialize(stream);
    stream.Close();

    WriteLine($”{o.id}, {o.name}, {o.salary}”); //-> 1 Maxi 0
  } }
```