C#
Serialisierung

70-483 – Skill 4.4

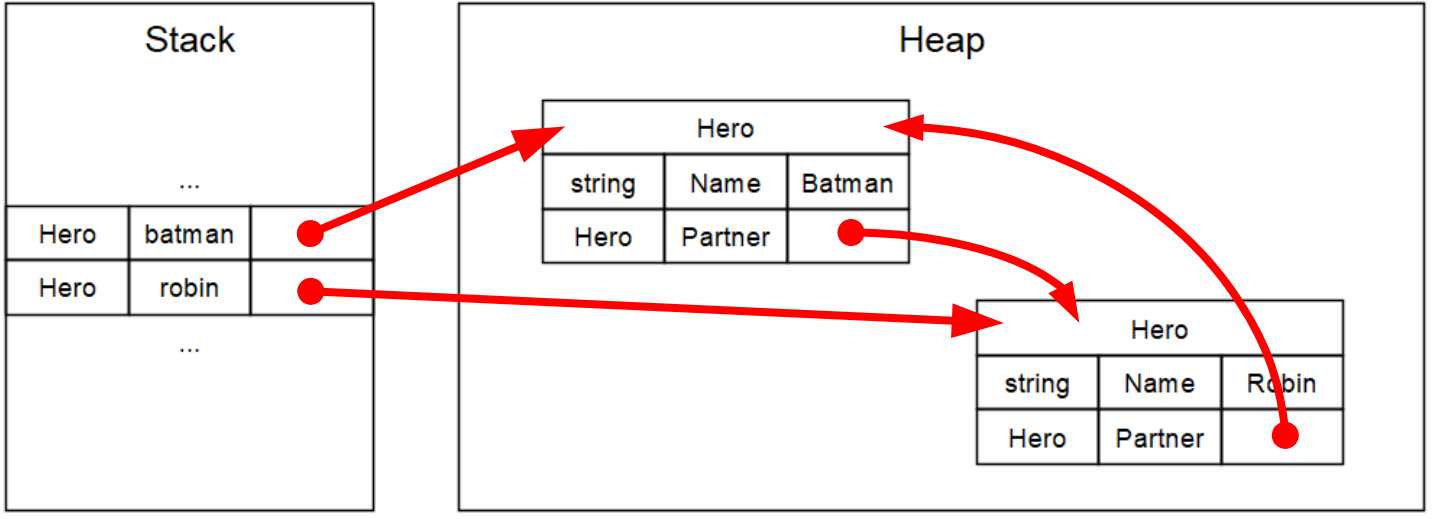
Grundlagen

- Serialisierung beschreibt den Prozess, komplexe Objekte in ein flaches Text- oder Byteformat zu wandeln.
- Dabei wird der komplette Zustand eines Objekts, inklusive aller referenzierten Objekte in einen Datenstrom gewandelt.
- Dadurch wird es möglich, Objekte auf einem Datenträger zu speichern oder sie über ein Netzwerk zu übertragen.
- Der umgekehrte Prozess, bei dem aus dem Text- oder Byteformat wieder Objekte erzeugt werden, nennt sich Deserialisierung.
- Bei der Serialisierung muss ein klar definiertes Format erzeugt werden, damit eine Deserialiserung erfolgen kann.

Zirkuläre Referenzen

```
class Hero {
    public string Name;
    public Hero? Partner;

    public Hero(string name) {
        Name = name;
    }
}
```

```
static void Main(string[] args) {
    Hero batman = new Hero("Batman");
    Hero robin = new Hero("Robin");

    batman.Partner = robin;

    robin.Partner = batman;
}
```



Serialisierungsverfahren

- Grundsätzlich stehen im .NET-Framework zwei Methoden zur Serialisierung und Deserialisierung zur Verfügung:
- `XmlSerializer`

  XML-Format
  zirkuläre Referenzen werden nicht unterstützt

- `JsonSerializer`

  JSON-Format
  zirkuläre Referenzen werden unterstützt

- Der ehemals vorhandene `BinaryFormatter` sollte unter keinen Umständen mehr genutzt werden.

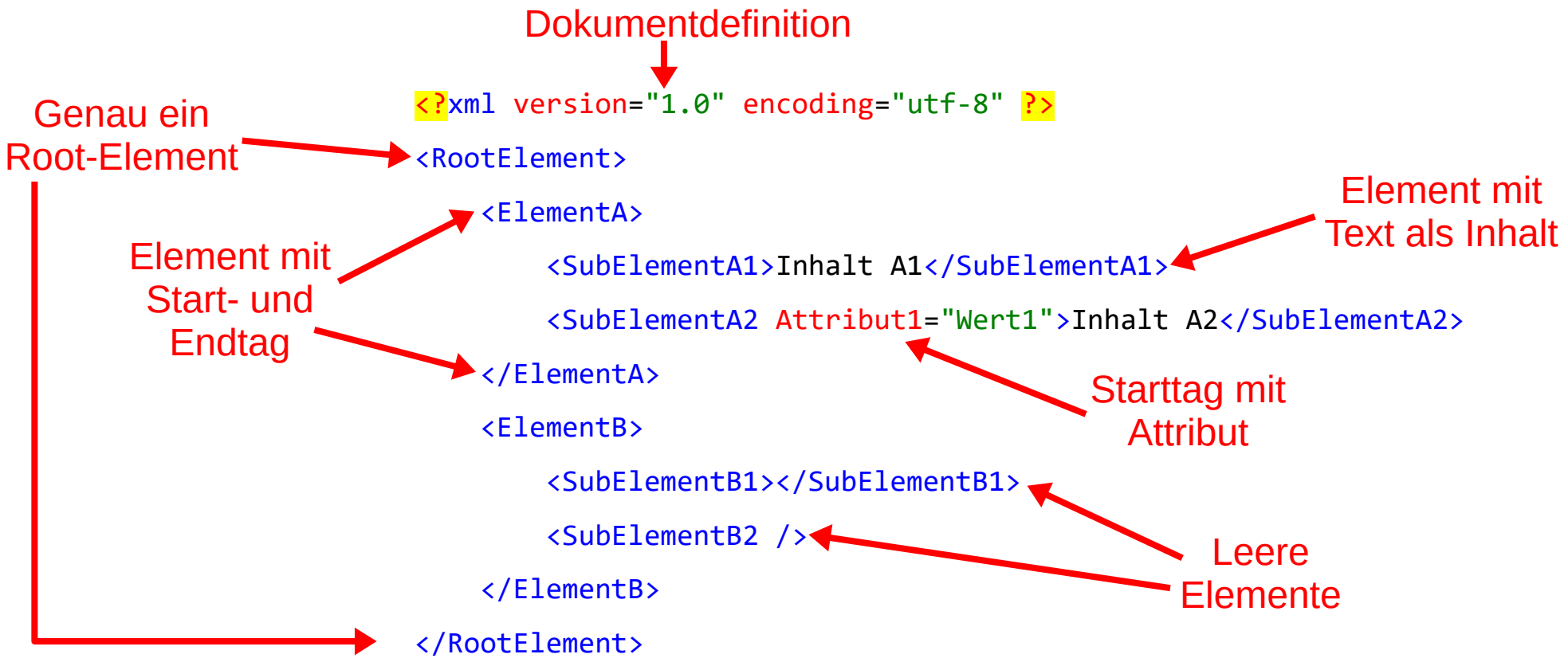
XML-Serialisierung

XML-Format

- Die Extensible Markup Language ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten als Textdatei.
- XML ist plattformunabhängig und sowohl für Menschen als auch für Maschinen lesbar.
- XML-Dokumente bestehen aus Elementen (sogenannten Knoten), die Text oder aber weitere Elemente enthalten können.
- Linus Torvalds soll über XML gesagt haben:

  “XML is crap. Really. There are no excuses. XML is nasty to parse for humans, and it’s a disaster to parse even for computers. There’s just no reason for that horrible crap to exist.”

XML-Format


Dokumentdefinition
Genau ein Root-Element
Element mit Start- und Endtag
Element mit Text als Inhalt
Starttag mit Attribut
Leere Elemente
<?xml version="1.0" encoding="utf-8" ?>
<RootElement>
<ElementA>
<SubElementA1>Inhalt A1</SubElementA1>
<SubElementA2 Attribut1="Wert1">Inhalt A2</SubElementA2>
</ElementA>
<ElementB>
<SubElementB1></SubElementB1>
<SubElementB2 />
</ElementB>
</RootElement>

XmlSerializer

- Der `XmlSerializer` kann Objekte in eine XML-Struktur serialisieren und auch wieder deserialisieren.
- Dadurch sind die serialisierten Daten auch für Menschen lesbar.
- Dabei gibt es aber leider eine ganze Reihe von Einschränkungen:

  Die Klasse muss `public` deklariert sein.

  Es werden nur `public` Felder und Properties serialisiert.

  Die Properties müssen lesenden und schreiben Zugriff erlauben.

  Die Klasse muss einen parameterlosen `public`-Konstruktor haben.

  Der `XmlSerializer` muss explizit für einen Datentyp aufgerufen werden.

XmlSerializer

- Die Serialisierung mit dem `XmlSerializer` funktionert grundsätzlich recht einfach.
- Das `XmlSerializer`-Objekt ist immer auf einen speziellen Typ festgelegt.
- Das mehrfache Serialisieren in die gleiche Datei funktioniert bei XML nicht. Hier muss mit Collections gearbeitet werden.

```
static void XmlExample() {
    Hero batman = new Hero("Batman", 42);
    batman.Stuff = "XYZ";
    batman.Partner = new Hero("Robin", 24);

    XmlSerializer xml =
        new XmlSerializer(typeof(Hero));

    using (FileStream stream =
        File.Create("Batman.xml")) {
        xml.Serialize(stream, batman);
    }

    Hero batmanCopy;
    using (FileStream stream =
        File.OpenRead("Batman.xml")) {
        batmanCopy =
            xml.Deserialize(stream) as Hero;
    }
    Console.WriteLine(batmanCopy.Name);
    Console.WriteLine(batmanCopy.GetAge());
}
```

Attribute für XmlSerializer

- Die Serialisierung kann über Attribute in der zu serialisierenden Klasse gesteuert werden:

`[XmlRoot]`	Legt den Namen des Root-Elements fest
`[XmlElement]`	Steuert die Ausgabe als XML-Element
`[XmlAttribute]`	Steuert die Ausgabe als XML-Attribut
`[XmlIgnore]`	Wird bei der Ausgabe übergangen
`[XmlArray]`	Steuert die Ausgabe als Array im XML
`[XmlArrayItem]`	Steuert die Ausgabe eines Elements im Array

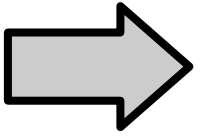
...

https://docs.microsoft.com/de-de/dotnet/standard/serialization/attributes-that-control-xml-serialization

Attribute für XmlSerializer

```
public class Hero {

    [XmlElement(Order = 4)]
    public DateTime DateOfBirth;

    [XmlArray("ListOfGadgets", Order = 3)]
    [XmlArrayItem("Gadget")]
    public List<string>? Gadgets { get; set; }

    [XmlElement("RealNameFirst", Order = 1)]
    public string? FirstName { get; set; }

    [XmlElement("RealNameLast", Order = 2)]
    public string? LastName { get; set; }

    [XmlAttribute("Name")]
    public string? HeroName { get; set; }

    [XmlAttribute("BelongsTo")]
    public string? Gang { get; set; }

    public Hero() {}
}
```

```
static void Main(string[] args) {
    Hero batman = new Hero() {
        // Batman details
    };
    Hero ironman = new Hero() {
        // Iron Man details
    };

    List<Hero> list = new List<Hero>();
    list.Add(batman);
    list.Add(ironman);

    XmlSerializer xml =
        new XmlSerializer(typeof(List<Hero>),
        new XmlRootAttribute("Heroes"));

    xml.Serialize(Console.Out, list);
}
```



Attribute für XmlSerializer

```
<?xml version="1.0" encoding="ibm850"?>
<Heroes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Hero Name="Batman" BelongsTo="Justice League">
    <RealNameFirst>Bruce</RealNameFirst>
    <RealNameLast>Wayne</RealNameLast>
    <ListOfGadgets>
      <Gadget>Batmobile</Gadget>
      <Gadget>Batsuit</Gadget>
      <Gadget>Batsomething</Gadget>
    </ListOfGadgets>
    <DateOfBirth>1978-11-11T00:00:00</DateOfBirth>
  </Hero>
  <Hero Name="Iron Man" BelongsTo="Avengers">
    <RealNameFirst>Tony</RealNameFirst>
    <RealNameLast>Stark</RealNameLast>
    <ListOfGadgets>
      <Gadget>Suit</Gadget>
      <Gadget>Money</Gadget>
      <Gadget>Jarvis</Gadget>
    </ListOfGadgets>
    <DateOfBirth>1976-09-09T00:00:00</DateOfBirth>
  </Hero>
</Heroes>
```

JSON-Serialisierung

JSON-Format

- JSON (JavaScript Object Notation) ist ein kompaktes textbasiertes und für den Menschen lesbares Datenformat.
- JSON wurde 1997 von Douglas Crockford entwickelt.
- JSON wird zur Speicherung und Übertragung von Daten zwischen Anwendung genutzt.
- Wie bei XML ist es auch mit JSON möglich, verschachtelte Datenstrukturen zu erzeugen.
- JSON ist aber deutlich leichtgewichtiger als XML.
- Und JSON ist leichter lesbar als XML.

JSON-Format

- JSON-Daten sind als Schlüssel-Wert-Paare organisiert.
- Die Schlüssel stehen am Anfang in doppelten Anführungszeichen, die Werte stehen mit einem Doppelpunkt getrennt dahinter.
- Es werden vier Typen von einfachen Werten unterschieden:

Zeichenkette	`"name": "Georg"`
Zahl	`"alter": 47`
Wahrheitswert	`"verheiratet": false`
Nullwert	`"beruf": null`

- Die Schlüssel-Wert-Paare müssen jetzt noch organisiert werden.

JSON-Format

- Objekte werden mit geschweiften Klammern eingerahmt und fassen mehrere, unsortierte Schlüssel-Wert-Paare zusammen.
- Dabei können Objekte auch weitere Objekte enthalten:

```
{
  "name": "Georg",
  "verheiratet": false,
  "alter": 47,
  "adresse": {
    "strasse": "Zum Beispiel 1 a",
    "plz": "12345",
    "Ort": "Musterstadt"
  },
  "beruf": null
}
```

JSON-Format

- Arrays werden in eckige Klammern eingerahmt und fassen mehrere unsortierte Werte des selben Typs zusammen.
- Einzelwerte müssen vom selben Typ sein, dabei kann es sich aber auch um den Typ Objekt handeln.
- Der innere Objekt-Aufbau kann unterschiedlich sein.
- Jedes JSON-File beginnt mit einem Objekt oder Array.

```
{
  "name": "Georg",
  "kinder": [
    {
      "name": "Georgina",
      "alter": 5
    },
    {
      "name": "Georg 2",
      "alter": 7
    }
  ],
  "hobbies": ["Lesen", "Sport", "BWL"]
}
```

Vergleich XML / JSON

```xml
<person>
  <name>Georg</name>
  <alter>47</alter>
  <verheiratet>false</verheiratet>
  <beruf xsi:nil="true"/>
  <kinder>
    <person>
      <name>Lukas</name>
      <alter>19</alter>
      <schulabschluss>Gymnasium</schulabschluss>
    </person>
    <person>
      <name>Lisa</name>
      <alter>14</alter>
      <schulabschluss xsi:nil="true"/>
    </person>
  </kinder>
</person>
```

XML
301 Zeichen

```json
{
  "name": "Georg",
  "alter": 47,
  "verheiratet": false,
  "beruf": null,
  "kinder": [
    {
      "name": "Lukas",
      "alter": 19,
      "schulabschluss": "Gymnasium"
    },
    {
      "name": "Lisa",
      "alter": 14,
      "schulabschluss": null
    }
  ]
}
```

JSON
177 Zeichen

JsonSerializer

- Mit dem `JsonSerializer` können Objekte in das JSON-Format serialisiert werden.
- Das JSON-Format ist platzsparend und wird häufig z. B. zum Datenaustausch zwischen Webserver und Client eingesetzt.
- Der `JsonSerializer` kann ab Framework 4.7.2 bzw. Core 3.0 über den Namespace `System.Text.Json` eingebunden werden.
- Zuvor musste man diesen als NuGet-Package des Drittanbieters Newtonsoft einbinden.
- Der `JsonSerializer` kann auch Zirkelbezüge darstellen, obwohl die Ausgabe später für Menschen lesbar ist.

JsonSerializer

```
class Hero {
    public string? HeroName
        { get; set; }

    [JsonIgnore]
    public string? FirstName
        { get; set; }

    [JsonIgnore]
    public string? LastName
        { get; set; }

    public string? Gang
        { get; set; }

    public Hero? Partner
        { get; set; }

    public List<string>? Gadgets
        { get; set; }
}
```

```
static void Main(string[] args) {
    Hero batman = new Hero() {
        HeroName = "Batman",
        FirstName = "Bruce",
        LastName = "Wayne",
        Gang = "Justice League",
        Gadgets = new List<string>()
            { "Batmobile", "Batsuit", "Batarang" }
    };

    string batmanJson = JsonSerializer.Serialize(batman);

    Console.WriteLine(batmanJson);

    Hero newBatman =
        JsonSerializer.Deserialize<Hero>(batmanJson);

    Console.WriteLine(newBatman.HeroName);
}
```