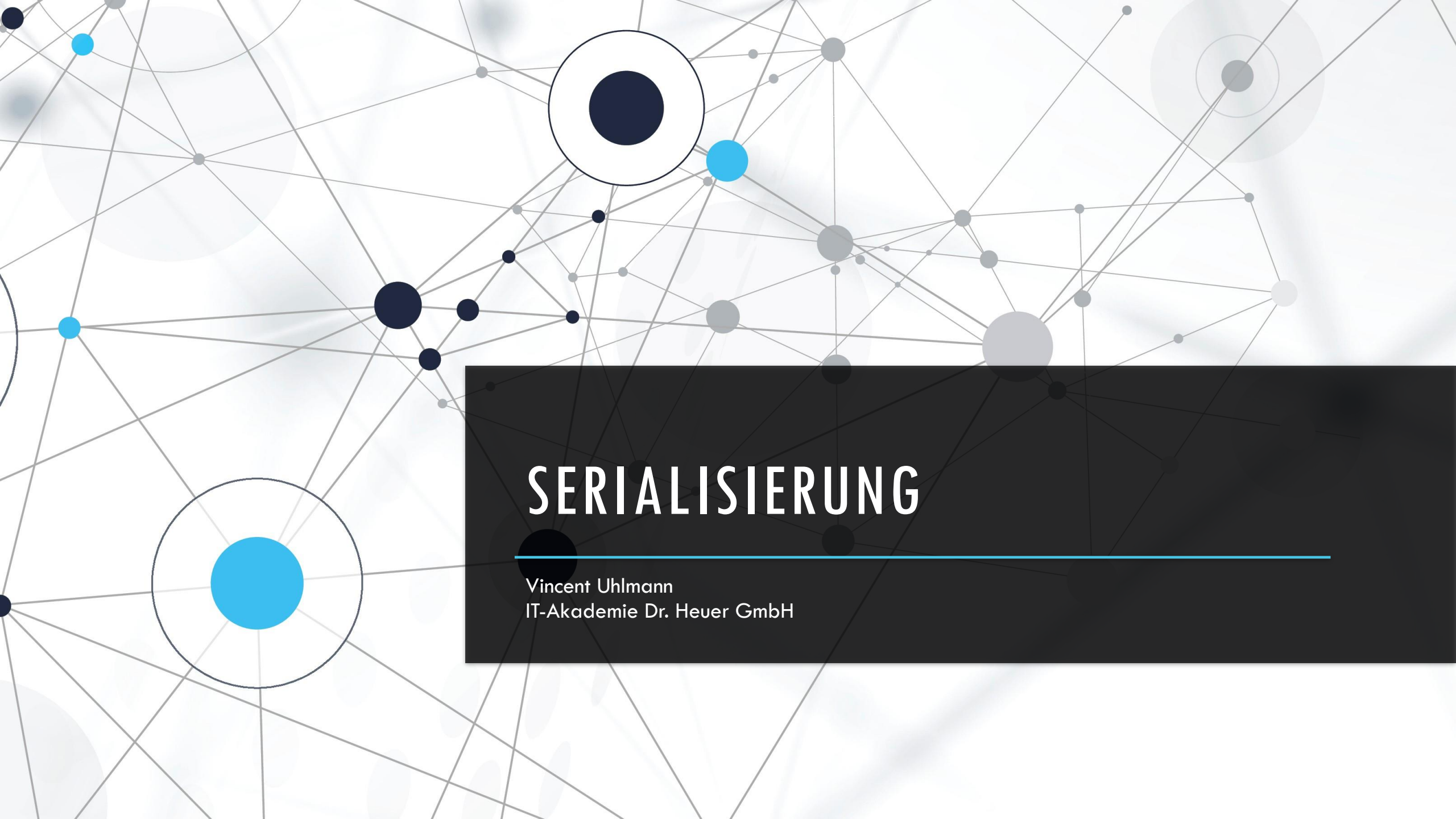


SERIALISIERUNG

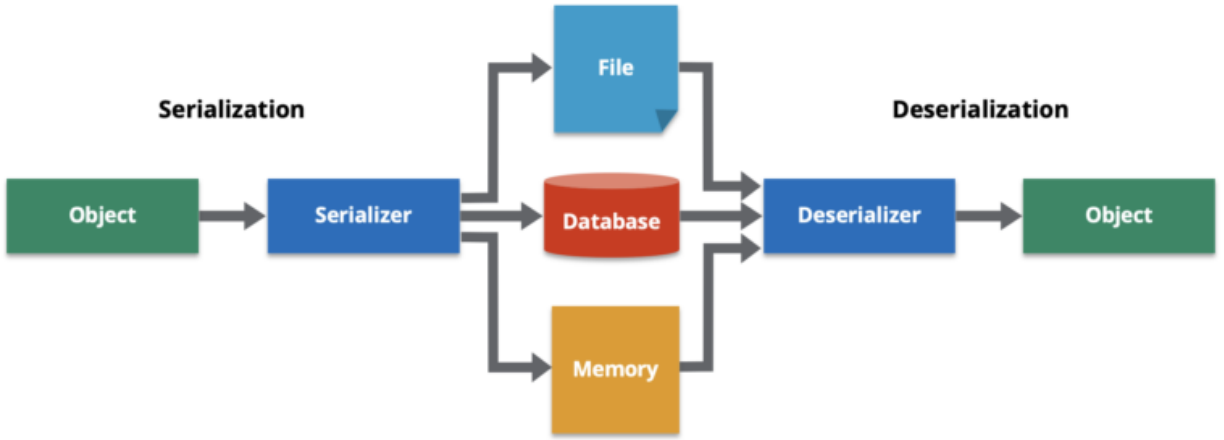
Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

WAS IST SERIALISIERUNG?

- Serialisierung ist der Prozess der Umwandlung von Objekten in eine andere Form
- Dient z.B. der Speicherung oder Übertragung von Daten
- Der Zustand eines Objektes wird inklusive aller referenzierten Objekte in ein anderes Format umgewandelt
- Nach der Serialisierung haben Änderungen am eigentlichen Objekt keine Auswirkungen mehr auf das serialisierte Objekt
- Die Umkehrung der Serialisierung wird als Deserialisierung bezeichnet

WOFÜR?

- Serialisierung wird genutzt, um Objekte in z.B. Dateien zu speichern und zu einem späteren Zeitpunkt wiederherzustellen
- Für die Übertragung von Objekten zwischen Systemen z.B. über ein Netzwerk müssen die Daten vorher Serialisiert werden



SERIALISIERUNGSVERFAHREN

- Es gibt definierte Formate, die festlegen, wie Daten serialisiert oder deserialisiert werden
- Diese gewährleisten eine einheitliche Interpretation der Daten
- Dazu gehören:

- XML (Extensible Markup Language)
- JSON (JavaScript Object Notation)
- Protocol Buffers
- FlatBuffers
- …

XML (EXTENSIBLE MARKUP LANGUAGE)

- XML ist ein Format zur Darstellung strukturierter Daten
- Die XMLSerializer Klasse ermöglicht die XML-Serialisierung von Objekten
- Namespace: System.Xml.Serialization

- Nur öffentliche Felder/Eigenschaften werden serialisiert
- Die zu serialisierende Klasse muss öffentlich sein
- Es muss ein Konstruktor ohne Parameter vorhanden sein (kann auch privat sein)

XML DATENSTRUKTUR

- In XML können einfache Datentypen wie string, int, double, bool usw. direkt serialisiert werden
- Für komplexe Datentypen, wie z.B. Klassen und Strukturen, können XML-Elemente und -Attribute verwendet werden, um die Struktur der Daten darzustellen
- Sammlungen wie List<T>, Dictionary<TKey, TValue> können ebenfalls serialisiert werden, wobei die Elemente der Sammlung als XML-Elemente dargestellt werden

```
<?xml version="1.0" encoding="UTF-8"?>
<Person xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Ints>
    <int>1</int>
    <int>2</int>
  </Ints>
  <Name>Peter</Name>
  <Address>Hansweg 12</Address>
</Person>
```

XML

```
public class Person
{
    public string Name { get; set; }
    public string Address { get; set; }

    private Person()
    {
    }

    public Person(string name, string address)
    {
        Name = name;
        Address = address;
    }
}
```

XML IN C#

- In C# wird die Serialisierung von XML durch die XmlSerializer-Klasse unterstützt

```
// Serialisierung
Person person = new Person("Peter","Bachweg 4");

XmlSerializer serializer = new XmlSerializer(typeof(Person));

using TextWriter writer = new StreamWriter(@"C:\Users\vu\Desktop\person.xml");

serializer.Serialize(writer, person);
```

XML IN C#

- In C# wird die Deserialisierung von XML ebenfalls durch die XmlSerializer-Klasse unterstützt

```
// Deserialisierung
XmlSerializer deserializer = new XmlSerializer(typeof(Person));

using TextReader reader = new StreamReader(@"C:\Users\vu\Desktop\person.xml");

Person? person = (Person?)deserializer.Deserialize(reader);

if(person != null)
    Console.WriteLine(person.Name + " " + person.Address); // Peter Bachweg 4
```

XML-ATTRIBUTE

- XML bietet Attribute, mit denen sich das Verhalten der Serialisierung und Deserialisierung konfigurieren lässt
- Das XmlElement-Attribut kann verwendet werden, um eine Eigenschaft als XML-Element zu serialisieren (Standard)
- Das XmlAttribute-Attribut sorgt zum Beispiel dafür, dass eine Eigenschaft als XML-Attribut und nicht als Element serialisiert wird
- Das Attribut XmlIgnore schließt eine Eigenschaft von der Serialisierung aus
- …

- Eine Liste aller möglichen Attribute kann hier gefunden werden
- https://learn.microsoft.com/de-de/dotnet/standard/serialization/attributes-that-control-xml-serialization

XMLATTRIBUTE-ATTRIBUT

- Das XmlAttribute-Attribut sorgt dafür, dass eine Eigenschaft als XML-Attribut und nicht als Element serialisiert wird

```
public class Person
{
    [XmlAttribute]
    public string Name { get; set; } = string.Empty;

    public string Address { get; set; } = string.Empty;
...
```

```
<?xml version="1.0" encoding="utf-8"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema" Name="Peter">
  <Address>Bachweg 4</Address>
</Person>
```

XMLIGNORE-ATTRIBUT

- Das XmlIgnore-Attribut schließt eine Eigenschaft von der Serialisierung aus

```
public class Person
{
    [XmlIgnore]
    public string Name { get; set; } = string.Empty;

    public string Address { get; set; } = string.Empty;
...
```

```
<?xml version="1.0" encoding="utf-8"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Address>Bachweg 4</Address>
</Person>
```

XMLTEXT-ATTRIBUT

- Das XmlText-Attribut sorgt dafür, dass eine Eigenschaft als XML-Text serialisiert wird

```
public class Person
{
    public Car Car { get; set; } = new Car { Name = "Audi" };
}

public class Car
{
    [XmlText]
    public string Name { get; set; }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Car>Audi</Car>
</Person>
```

JSON (JAVASCRIPT OBJECT NOTATION)

- JSON ist ein Datenformat für den Austausch von Daten zwischen Anwendungen
- Die Daten liegen in lesbarer Textform vor
- JSON ist Sprachunabhängig und für fast jede Sprache sind Parser (zum Serialisieren und Deserialisieren) vorhanden
- Schneller und einfacher verständlich als XML

JSON DATENSTRUKTUR

- Objekte: Eine Sammlung von Schlüssel-Wert-Paaren die in geschweiften Klammern {} eingefasst sind
- Arrays: Eine Sammlung von Werten, die in eckigen Klammern [] eingefasst sind
- Werte: Können Strings, Zahlen, Objekte, Arrays, Booleans oder null sein

- Beispiel eines JSON Objektes

```
{
   "employee":
   {
      "name":"John",
      "age":30,
      "city":"New York"
   }
}
```

JSON DATENSTRUKTUR

- Beispiel eines JSON Objektes mit Array

```
{
   "employees":[
      {
         "name":"John",
         "age":30,
         "city":"New York"
      },
      {
         "name":"John",
         "age":30,
         "city":"New York"
      }
   ]
}
```

JSON IN C#

- In C# gibt es verschiedene Möglichkeiten, Objekte in JSON zu serialisieren oder von JSON zu deserialisieren
- .NET bzw. .NET Core enthält eine Klasse namens JsonSerialiser, die im Namespace System.Text.Json liegt
- Da sie im klassischen .NET-Framework nicht existierte, haben sich JSON-Parser von anderen Anbietern wie z.B. Newtonsoft durchgesetzt
- Newtonsoft.Json (NuGet Packet) ist eine der am häufigsten verwendeten Bibliotheken für JSON-Serialisierung
- Bietet eine einfache und leistungsstarke API

NEWTONSOFT.JSON

- Installation via NuGet (Newtonsoft.Json)
- Namespace: using Newtonsoft.Json

- Um ein Objekt in JSON zu serialisieren oder deserialiseren, wird die Klasse JsonConvert verwendet
- Hinweis: Standardmäßig werden nur Eigenschaften und Felder, die als öffentlich (public) deklariert sind, serialisiert

- Die Methode SerializeObject serialisiert das übergebene Objekt und gibt einen String zurück
- Die Methode DeserializeObject deserialisiert den übergebenen String und gibt das Objekt zurück

NEWTONSOFT.JSON

```
public class Person
{
    public string Name { get; set; }
    public string Address { get; set; }

    public Person(string name, string address)
    {
        Name = name;
        Address = address;
    }
}
```

NEWTONSOFT.JSON

```
// Serialisierung
Person person = new Person("Peter", "Bauerweg 2");

string json = JsonConvert.SerializeObject(person);

Console.WriteLine(json); // {"Name":"Peter","Address":"Bauerweg 2"}


// Deserialisierung
Person originalPerson = JsonConvert.DeserializeObject<Person>(json);
Console.WriteLine(originalPerson.Name); // Peter
```

NEWTONSOFT.JSON ATTRIBUTE

- Newtonsoft.Json bietet Attribute, die verwendet werden können, um das Verhalten der Serialisierung und Deserialisierung zu konfigurieren
- Das JsonProperty-Attribut kann zum Beispiel verwendet werden, um den Namen anzupassen
- Das JsonIgnore-Attribut kann verwendet werden, um eine Eigenschaft zu ignorieren
- …

- Eine Liste aller möglichen Attribute kann hier gefunden werden
- https://www.newtonsoft.com/json/help/html/serializationattributes.htm

JSONPROPERTY-ATTRIBUT

- Das JsonProperty-Attribut kann verwendet werden, um die Eigenschaften während der Serialisierung und Deserialisierung umzubenennen

```
public class Person
{
    [JsonProperty("name")]
    public string Name { get; set; }

    [JsonProperty("address")]
    public string Address { get; set; }

    public Person(string name, string address)
    {
        Name = name;
        Address = address;
    }
}
```

```
{"name":"Peter","address":"Bauerweg 2"}
```

JSONIGNORE-ATTRIBUT

- Das JsonIgnore-Attribut kann verwendet werden, um Eigenschaften während der Serialisierung und Deserialisierung zu ignorieren

```
public class Person
{
    [JsonProperty("name")]
    public string Name { get; set; }

    [JsonIgnore]
    public string Address { get; set; }

    public Person(string name, string address)
    {
        Name = name;
        Address = address;
    }
}
```

```
{"name":"Peter"}
```

JSON VS XML

```
{"Name":"Peter","Address":"Bachweg 4"}
```

```
<?xml version="1.0" encoding="utf-8"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Name>Peter</Name>
  <Address>Bachweg 4</Address>
</Person>
```