SERIALIZATION IN .NET

softserve

AGENDA

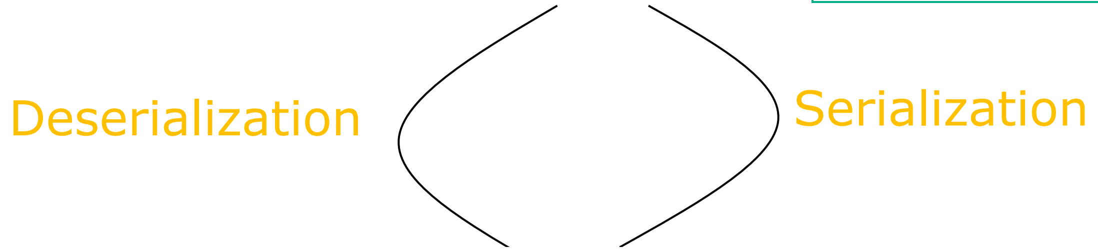
- What is Serialization?
- Serialization in .NET
- Binary serialization
- XML Serialization in C#
- Serialization in JSON format

What is Serialization?

- **Serialization** is the process of transforming an object or object graph that you have in-memory into a stream of bytes or text.
- **Deserialization** is the opposite. You take some bytes or text and transform them into an object.

```
[Serializable]
public class Person
 {
…
 }
```

```
Person st1 = new Person();
 st1.FirstName = "Iryna";
 st1.LastName = "Koval";
 st1.BirthDate = new DateTime(1981, 8, 17);
```



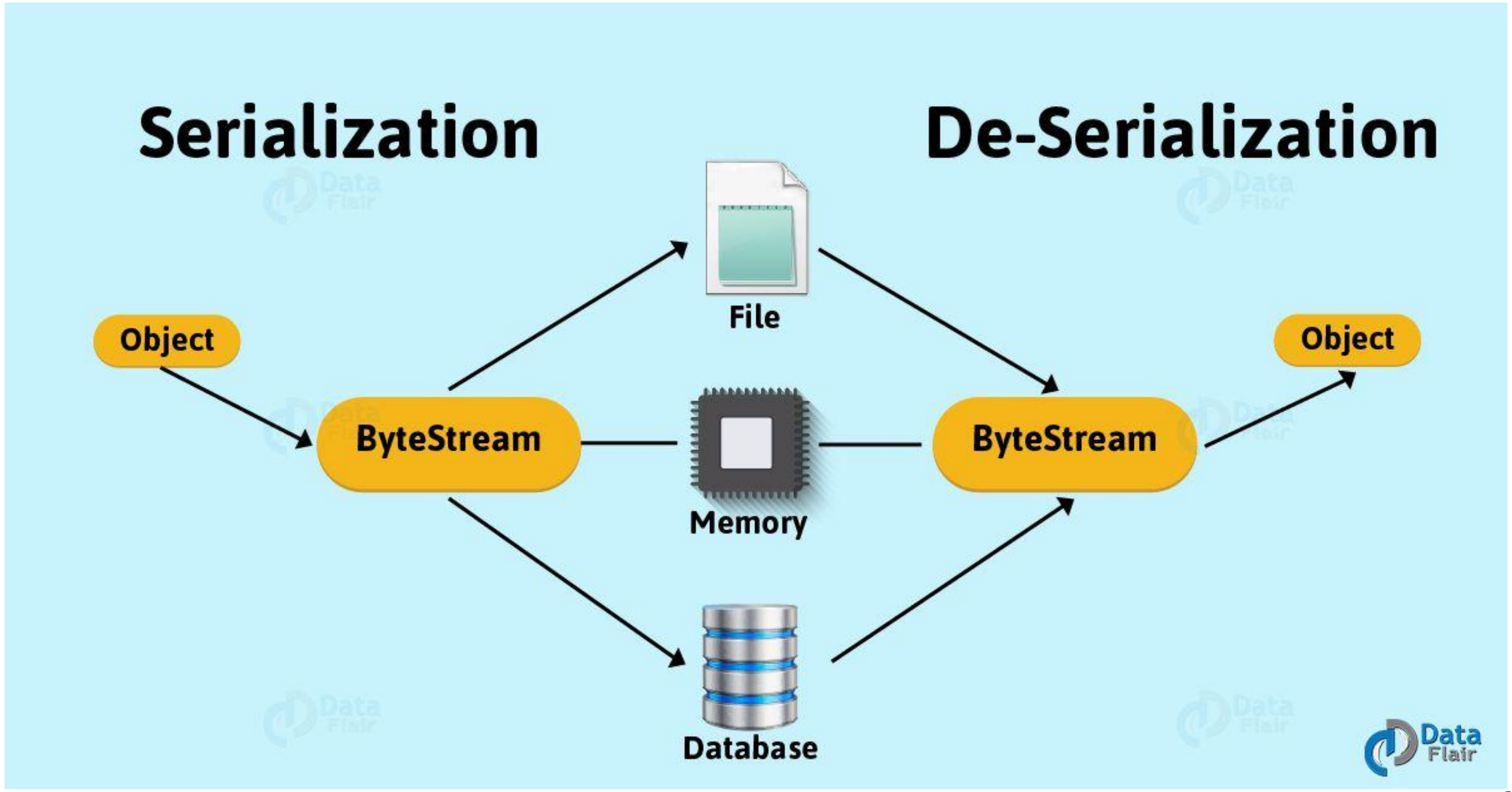
```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C 54 65
73 74 A0 0C 34 00 FE B1 DD F9 02 00 02 42 00 05
63 6F 75 6E 74 42 00 07 76 65 72 73 69 6F 6E 78
70 00 64
```

Serialization in .NET

- .NET Framework has classes (in the ***System.Runtime.Serialization*** and ***System.Xml.Serialization*** namespaces) that support:
  - ✔ ***binary,***
  - ✔ ***XML,***
  - ✔ ***JSON,***
  - ✔ ***own custom serialization.***

- The .NET Framework offers three serialization mechanisms that you can use by default:
  - ✔ ***BinaryFormatter***
  - ✔ ***XmlSerializer***
  - ✔ ***DataContractSerializer***

Serialization in .NET



softserve

Binary serialization

- In **binary** serialization all items are serialized, **even private field and read-only**, increasing productivity.
- In **binary** serialization, there is used a binary encoding to provide a compact object serialization for storage or transmission in a network flows based on sockets.
- It is not suitable for data transmission through the firewall, but provides better performance while saving data.
- namespace **System.Runtime.Serialization.Formatters.Binary**
- classes **BinaryFormatter** and **SoapFormatter** .

BinaryFormatter

```
[Serializable]
class Person
{
  private int _id;
  public string FirstName;
  public string LastName;
  public void SetId(int id)
  {
      _id = id;
  }
}
```

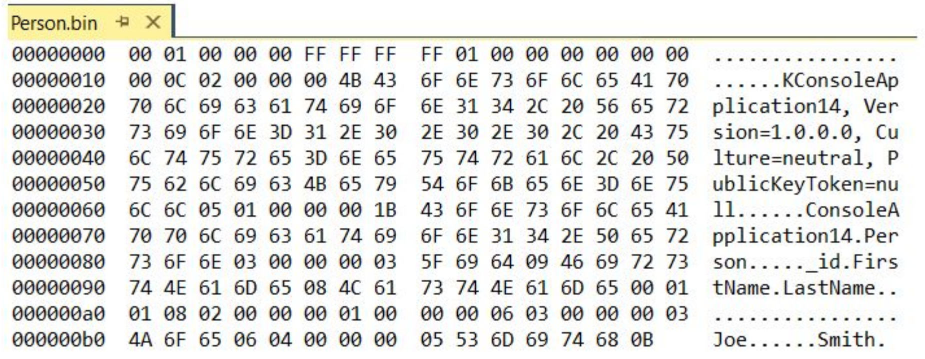
```
Person person = new Person();
person.SetId(1);
person.FirstName = "Joe";
person.LastName = "Smith";

IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("Person.bin",
FileMode.Create, FileAccess.Write,
FileShare.None);

formatter.Serialize(stream, person);
stream.Close();
```

```
stream = new FileStream("Person.bin",
FileMode.Open, FileAccess.Read, FileShare.Read);

Person person2 =
(Person)formatter.Deserialize(stream);
stream.Close();
```



```
00000000  00 01 00 00 00 FF FF FF  FF 01 00 00 00 00 00 00  ................
00000010  00 0C 02 00 00 00 4B 43  6F 6E 73 6F 6C 65 41 70  ......KConsoleAp
00000020  70 6C 69 63 61 74 69 6F  6E 31 34 2C 20 56 65 72  plication14, Ver
00000030  73 69 6F 6E 3D 31 2E 30  2E 30 2E 30 2C 20 43 75  sion=1.0.0.0, Cu
00000040  6C 74 75 72 65 3D 6E 65  75 74 72 61 6C 2C 20 50  lture=neutral, P
00000050  75 62 6C 69 63 4B 65 79  54 6F 6B 65 6E 3D 6E 75  ublicKeyToken=nu
00000060  6C 6C 05 01 00 00 00 1B  43 6F 6E 73 6F 6C 65 41  ll......ConsoleA
00000070  70 70 6C 69 63 61 74 69  6F 6E 31 34 2E 50 65 72  pplication14.Per
00000080  73 6F 6E 03 00 00 00 03  5F 69 64 09 46 69 72 73  son....._id.Firs
00000090  74 4E 61 6D 65 08 4C 61  73 74 4E 61 6D 65 00 01  tName.LastName..
000000a0  01 08 02 00 00 00 01 00  00 00 06 03 00 00 00 03  ................
000000b0  4A 6F 65 06 04 00 00 00  05 53 6D 69 74 68 0B     Joe......Smith.
```

BinaryFormatter: Attributes

- ❖ To indicate that instances of this type can be serialized, mark it with the **[Serializable] attribute**. When you try to serialize the type that has no such attribute, a **SerializationException** occurs.
- ❖ If you do not want to serialize the fields within a class, apply the **[NonSerialized] attribute**.
- ❖ If a serializable class contains references to objects of other classes that are marked with a **[Serializable]** attribute, those objects are also serializable.
- ❖ the **[OptionalField]** attribute is used to make sure that the binary serializer knows that a field is added in a later version and that earlier serialized objects won’t contain this field

XMLSerializer

- ❖ The ***XmlSerializer*** (namespace **System.Xml.Serialization)** SOAP is a protocol for exchanging information with web services. It uses XML as the format for messages.
- ❖ XML is readable by both humans and machines, and it is independent of the environment it is used in.

- ❖ To **serialize** an object:
  - ✔ Create the object and set its public fields and properties.
  - ✔ Construct a **XmlSerializer** using the type of the object.
  - ✔ Call the **Serialize** method to generate either an XML stream or a file representation of the object's public properties and fields

- ❖ To **deserialize** an object:
  - ✔ Construct a **XmlSerializer** using the type of the object to deserialize.
  - ✔ Call the **Deserialize** method to produce a replica of the object. After deserialization you must cast the returned object to the type of the original

XMLSerializer: Attribute

❖ You can configure how the XmlSerializer serializes your type by using attributes. These attributes are defined in the *System.Xml.Serialization* namespace :

- ✔ ***XmlIgnore*** **-** can be used to make sure that an element is not serialized
- ✔ ***XmlAttribute*** **-** you can map a member to an attribute on its parent node.
- ✔ ***XmlElement*** **–** *by default*
- ✔ ***XmlArray*** **-** *is* used when serializing collections.
- ✔ ***XmlArrayItem*** **-** is used when serializing collections.

XMLSerializer

```
Person st1 = new Person();
st1.FirstName = "John";
st1.LastName = "Doe";
XmlSerializer xmlser = new XmlSerializer(typeof(Person));
Stream serialStream = new FileStream("person.xml", FileMode.Create);

xmlser.Serialize(serialStream, st1);
```

```
<?xml version="1.0"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <FirstName>John</FirstName>
  <LastName>Doe</LastName>
</Person>
```

```
serialStream = new FileStream("person.xml", FileMode.Open);

Person st2 = xmlser.Deserialize(serialStream) as Person;

Console.WriteLine(st2);
```

softserve

Complex and derived types serialization

```
[Serializable]
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}
```

```
[Serializable]
public class Order
{
     [XmlAttribute]
     public int ID { get; set; }

    [XmlIgnore]
     public bool IsDirty { get; set; }

     [XmlArray("Lines")]
     [XmlArrayItem("OrderLine")]
     public List<OrderLine> OrderLines { get;
set; }
}
```

```
[Serializable]
public class VIPOrder : Order
{
    public string Description { get; set;}
}
[Serializable]
public class OrderLine
{
    [XmlAttribute]
     public int ID { get; set; }

     [XmlAttribute]
     public int Amount { get; set; }

      [XmlElement("OrderedProduct")]
      public Product Product { get; set; }
}
```

```
[Serializable]
public class Product
{
    [XmlAttribute]
    public int ID { get; set; }

    public decimal Price { get; set; }
    public string Description { get; set; }}
```

softserve

Complex and derived types serialization

```
private static Order CreateOrder()
{
    Product p1 = new Product { ID = 1, Description = “p2”, Price = 9 };
    Product p2 = new Product { ID = 2, Description = “p3”, Price = 6 };
    Order order = new VIPOrder { ID = 4, Description = “Order for John Doe. Use the nice giftwrap”,
                                            OrderLines = new List<OrderLine> {
                                            new OrderLine { ID = 5, Amount = 1, Product = p1},
                                            new OrderLine { ID = 6 ,Amount = 10, Product = p2},
                                                                            }
                                           };
     return order;
}

XmlSerializer serializer = new XmlSerializer(typeof(Order), new Type[] { typeof(VIPOrder) });
string xml;
using (StringWriter stringWriter = new StringWriter())
{
     Order order = CreateOrder();
     serializer.Serialize(stringWriter, order);
      xml = stringWriter.ToString();
}
using (StringReader stringReader = new StringReader(xml))
    { Order o = (Order)serializer.Deserialize(stringReader);
        // Use the order}
```

JSON Serialization

- We can use **DataContractJsonSerializer** to serialize type instance to JSON string and deserialize JSON string to type instance
- **DataContractJsonSerializer** is under **System.Runtime.Serialization.Json** namespace.
- http://www.codeproject.com/Articles/272335/JSON-Serialization-and-Deserialization-in-ASP-NET#

DataContractJsonSerializer: Properties

DateTimeFormat - Gets the format of the date and time type items in object graph.

EmitTypeInformation - Gets or sets the data contract JSON serializer settings to emit type information.

IgnoreExtensionDataObject - Gets a value that specifies whether unknown data is ignored on deserialization and whether the IExtensibleDataObject interface is ignored on serialization.

KnownTypes - Gets a collection of types that may be present in the object graph serialized using this instance of the DataContractJsonSerializer.

MaxItemsInObjectGraph - Gets the maximum number of items in an object graph that the serializer serializes or deserializes in one read or write call.

SerializeReadOnlyTypes - Gets or sets a value that specifies whether to serialize read only types.

UseSimpleDictionaryFormat - Gets a value that specifies whether to use a simple dictionary format.

JSON Serialization. *class Person*

```
using System.Runtime.Serialization.Json;
```

```
    [DataContract]
    internal class Person
    {
        [DataMember]
        internal string name;

        [DataMember]
        internal int age;
    }
```

```
file.Position = 0;
Person p2 = (Person)ser.ReadObject(file);
Console.Write("Deserialized back, got
name={0}, age={1}", p2.name,p2.age);
```

```
{"age":42,"name":"John"}
```

```
 Person p = new Person();
 p.name = "John";
 p.age = 42;
 Stream file = new FileStream("person.json", FileMode.Create);
 DataContractJsonSerializer ser = new

DataContractJsonSerializer(typeof(Person));
ser.WriteObject(file, p);
```

softserve

Task 12

- For one of the previously developed classes, implement binary, xml and json serialization

Thank You

softserve