Java Programming
Module
Programming

DHBW Mannheim
Duale Hochschule
Baden-Württemberg

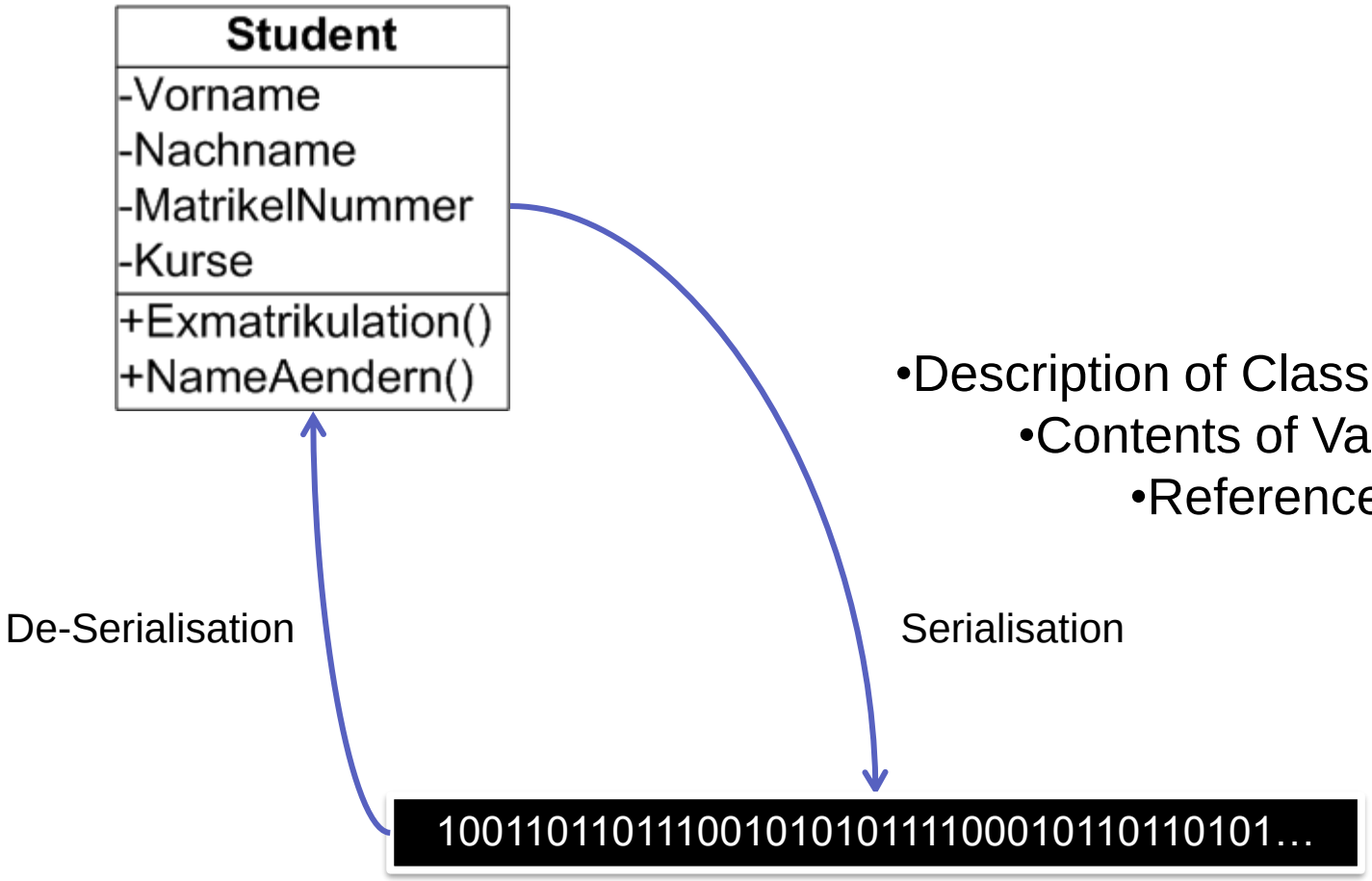
Prof. Dr. Holger D. Hofmann

Serialisation

- *Object Serialization supports the encoding of objects, and the objects reachable from them, into a stream of bytes* *JDK Documentation*

- Serialisation can
  - save the state of an Object at runtime
  - send Objects via a Network
  - transfer objects via the clipboard
  - realise a "deep copy" of Objects

Serialisation



Serialisation

- How to serialise an object
  - Standard Serialisation
  - XML Serialization using JavaBeans Persistence
  - XML JAXB Mapping (Java Architecture for XML Binding)
    - Generation of Java classes from XML Schema
    - „Data Binding“
    - No further need to use JAX (Simple API for XML) or DOM

Standard Realisation

- Class has to implement the interface Serializable
- Using the writeObject(Object) method, an object can be written to a **ObjectOutputStream** object
- Reading an Object via **readObject**() method

nothing to implement, but not "empty"

```
import java.io.Serializable; // "flag" interface
public class Student implements Serializable {
    String Name;
    String StudentID;
  transient int test; //this attribute is not serialised!
    ArrayList<Course> Courses;
}
```

Standard Serialisation

```
try { //Serialisation
FileOutputStream file = new FileOutputStream("data.txt");
ObjectOutputStream outStream = new ObjectOutputStream(file);
      outStream.writeObject(student);
 } catch (Exception e) {
    System.err.println( e);
 }


try { //De-Serialisation
      FileInputStream file = new FileInputStream("data.txt");
      ObjectInputStream inStream = new ObjectInputStream(file);
      Student student = (Student) inStream.readObject();
 } catch (Exception e) {
            System.err.println( e);
 }
```

Standard Serialisation

- Advantages
  - Easy-to-implement
- Disadvantages
  - Overhead
  - inflexible
  - Problems with versioning

Manual Serialisation

- Manuel writing and reading of object data
- Example:

```
private synchronized void writeObject( java.io.ObjectOutputStream s )
  throws IOException

private synchronized void readObject( java.io.ObjectInputStream s )
  throws IOException, ClassNotFoundException
```

Versioning

- Problem: objects of different class versions may be incompatible
- Solution
  - when implementing Serializable, a long constant ***serialVersionUID*** is implicitely created (can also manually be created as static final long serialVersionUID)
  - all changes to a class change ist SVUID
  - Trying to deserialize a wrong SVUID results in an InvalidClassException
  - command line programm serialver shows SVUID for a class
- When manually creating a serialVersionUID, the object itself has to guarantee for compatibility

Sending an Object over the Network

```
Socket s = new Socket( host, port );
OutputStream os = s.getOutputStream();
ObjectOutputStream oos = new
ObjectOutputStream( os );
oos.writeObject( object );
oos.flush();
```

[JI, 17.10.2]

Recursion

- Example: n! (factorial)
- Iterative solution?
- Recursive solution?

Is there a difference between OO and non-OO realisation?