Praktikum SW2

Input / Output

Streaming

Streams

Readers and Writers

Object serialization

Files

Summary

Streaming in Java

Package `java.io`

- byte-oriented input and output
- character-oriented input and output (for Unicode chars)

2 types of streams

- Byte-Streams
  - classes `InputStream-` und `OutputStream`
- Char-Streams:
  - classes `Reader-` und `Writer`

Streaming

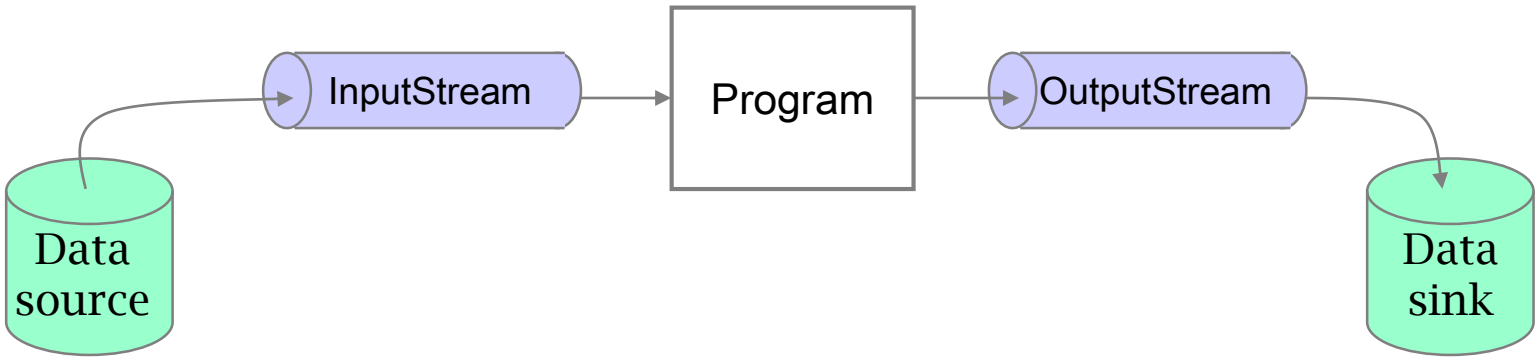
Stream of data into and out of program

Data will not be retrieved at once but stream in and out

Streams abstract from data sources and sinks

Streams abstrahieren von konkreten Quellen und Senken

- Files
- Sockets
- …



Base class InputStream

- Abstract base implementation of input streams
- Abstract method read(), which reads a single byte
- Concrete implementations of all other methods
- Methods work synchronously, i.e., they block until data available
- Methods throw `IOExceptions` when something works wrong

```
public abstract class InputStream {
   public abstract int read() throws IOException
   public int read(byte b[]) throws IOException
   public int read(byte b[], int off, int len)
       throws IOException
   public long skip(long n) throws IOException
   public int available() throws IOException
   public void close() throws IOException
   public synchronized void mark(int readlimit)
   public synchronized void reset() throws IOException
   public boolean markSupported()
}
```

Base class OutputStream

- Analogous to InputStream
- with abstract method write(), which writing a single byte
- and concrete write methods

```
public abstract class OutputStream {
    public abstract void write(int b) throws IOException;
    public void write(byte b[]) throws IOException
    public void write(byte b[], int off, int len) throws IOException
    public void flush() throws IOException
    public void close() throws IOException
}
```

```
public class IOException extends Exception {
    public IOException()
    public IOException(String s)
}
```

Implementation of Concrete Streams

As subclasses of `InputStream` and `OutputStream`

Overwrite methoden `read` and `write`

Example: `FileInputStream`: Reading from files

```
public class FileInputStream extends InputStream {
    public FileInputStream(String name) throws FileNotFoundException
    public FileInputStream(File file) throws FileNotFoundException
    public native int read() throws IOException
    …
}
```

native implementation!

Example: `FileOutputStream`: Writing to file

```
public class FileOutputStream extends OutputStream {
    public FileOutputStream(String name) throws FileNotFoundException
    public FileOutputStream(String name, boolean append)
                    throws FileNotFoundException
    public FileOutputStream(File file)
                    throws FileNotFoundException
    public native void write(int b) throws IOException
    …
}
```

native implementation!

Example: Copying a file

```
static void copyFile(String ifile, String ofile) {
    InputStream in = null;
    OutputStream out = null;
    try {
        in = new FileInputStream(ifile);
        out = new FileOutputStream(ofile);
        int b = in.read();
        while (b >= 0) {
            out.write(b);
            b = in.read();
        }
    } catch (FileNotFoundException e) {
        System.out.println("File does not exist");
    } catch (IOException e) {
        System.out.println("Error in reading or writing file");
    } finally {
       if (in != null) {
            try { in.close(); } catch (IOException ioe) {}
        }
        if (out != null) {
            try { out.close(); } catch (IOException ioe) {}
        }
    }
}
```

-1 for end of file

Note: Catching IOExceptions

Closing streams in finally block !!

TRY-WITH-RESOURCES

try-statement with resource management

- create and open resource
- use resource
- closing automatically

Resources must implement **AutoClosable**

```
try ( **** Create resources here **** )  {

      **** use resources here ****

} catch ( **** some exceptions **** e) {
     ...
}
```

Resources closed automatically

Example: Copying a file

Variant with try-with-resourced

```
static void copyFile_WithResources(String ifile, String ofile) {
    try (
            InputStream in  = new FileInputStream(ifile);
            OutputStream out = new FileOutputStream(ofile)
        ) {
        int b = in.read();
        while (b >= 0) {
            out.write(b);
            b = in.read();
        }
    } catch (FileNotFoundException e) {
        System.out.println("File does not exist");
    } catch (IOException e) {
        System.out.println("Error in reading or writing file");
    }
}
```

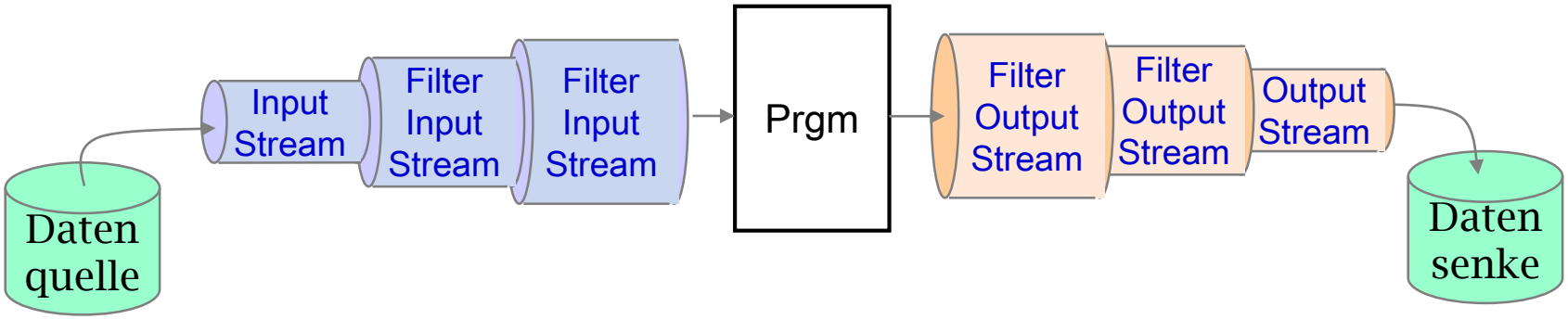
Closing of streams done automatically !!

FilterStreams: Pipelines of Streams

FilterStreams are streams which decorate other streams

Pipelines of streams allow

- base streams with new functionality
- advanced APIs




```
InputStream fin = new FileInputStream("input.txt");
BufferedInputStream bin = new BufferedInputStream(fin);
DataInputStream din = new DataInputStream(bin);
```

```
OutputStream fout = new FileOutputStream("output.txt");
BufferedOutputStream bout = new BufferedOutputStream(fout);
DataOutputStream dout = new DataOutputStream(bout);
```

FilterStreams: BufferedInputStream - BufferedOutputStream

Buffering input: `BufferedInputStream`

```
File ifile = new File("inputfile.txt");
InputStream in = new BufferedInputStream(
                          new FileInputStream(ifile));
```

Buffering output: `BufferedOutputStream`

- Output only when buffer full
- Use **flush()** for forcing output of buffer

```
File ofile = new File("outputfile.txt");
OutputStream out = new BufferedOutputStream(
                          new FileOutputStream(ofile));
```

FilterStreams: DataInputStream, DataOutputStream

Output of primitive data types

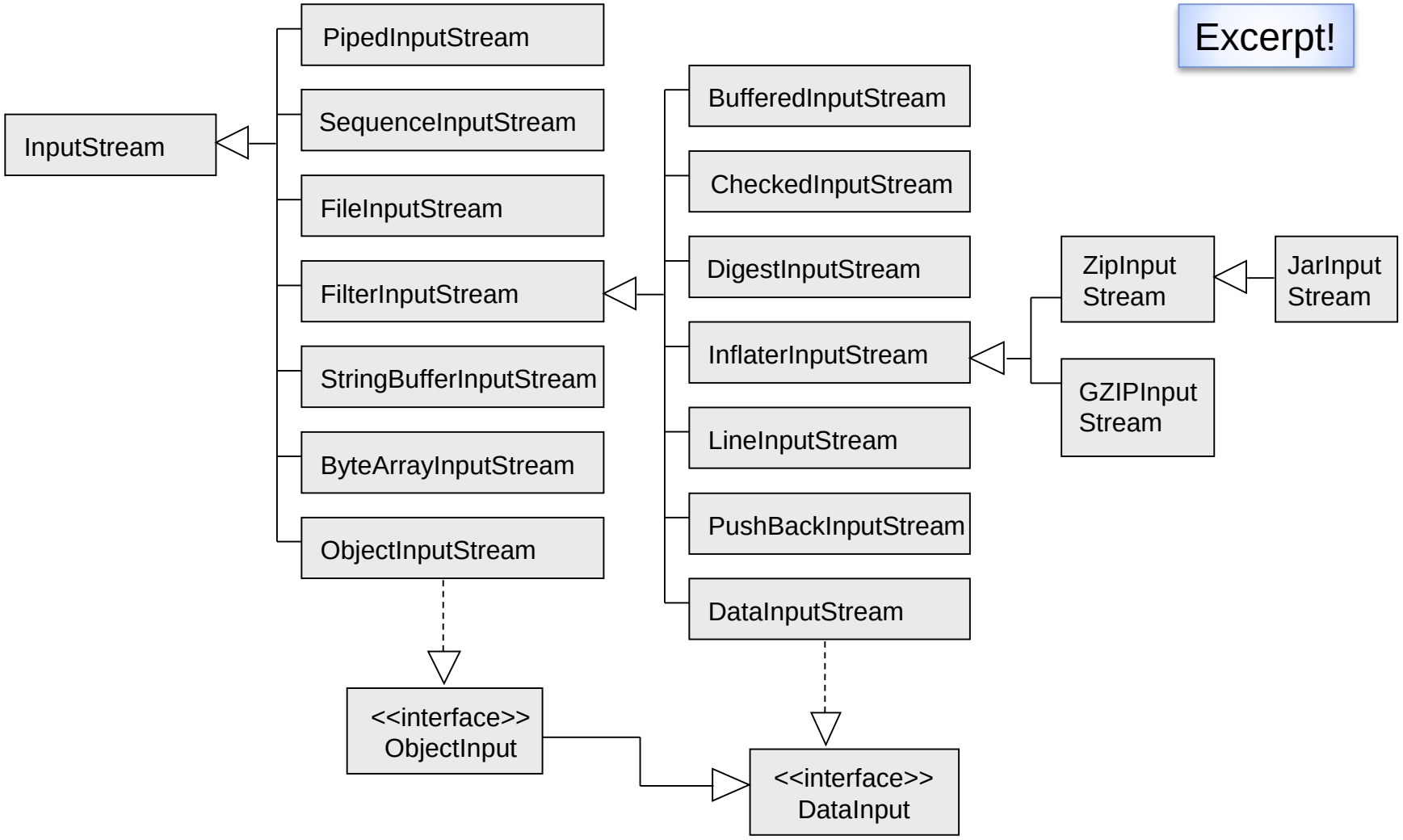
```
public class DataOutputStream
        extends FilterOutputStream {
  void writeBoolean(boolean v);
  void writeByte(int v);
  void writeBytes(String s);
  void writeChar(int v);
  void writeChars(String s);
  void writeDouble(double v);
  void writeFloat(float v);
  void writeInt(int v);
  void writeLong(long v);
  void writeShort(int v);
  void writeUTF(String str);
  ...
}
```

```
public class DataInputStream
       extends FilterInputStream {
  boolean readBoolean();
  byte    readByte();
  char    readChar();
  double  readDouble();
  float   readFloat();
  int     readInt();
  long    readLong();
  short   readShort();
  int     readUnsignedByte();
  int     readUnsignedShort();
  String  readUTF();
  ...
}
```

```
File fout = new File("output.data");
DataOutputStream out = new DataOutputStream(
        new FileOutputStream(fout));
out.writeDouble(Math.PI);
out.close();
```

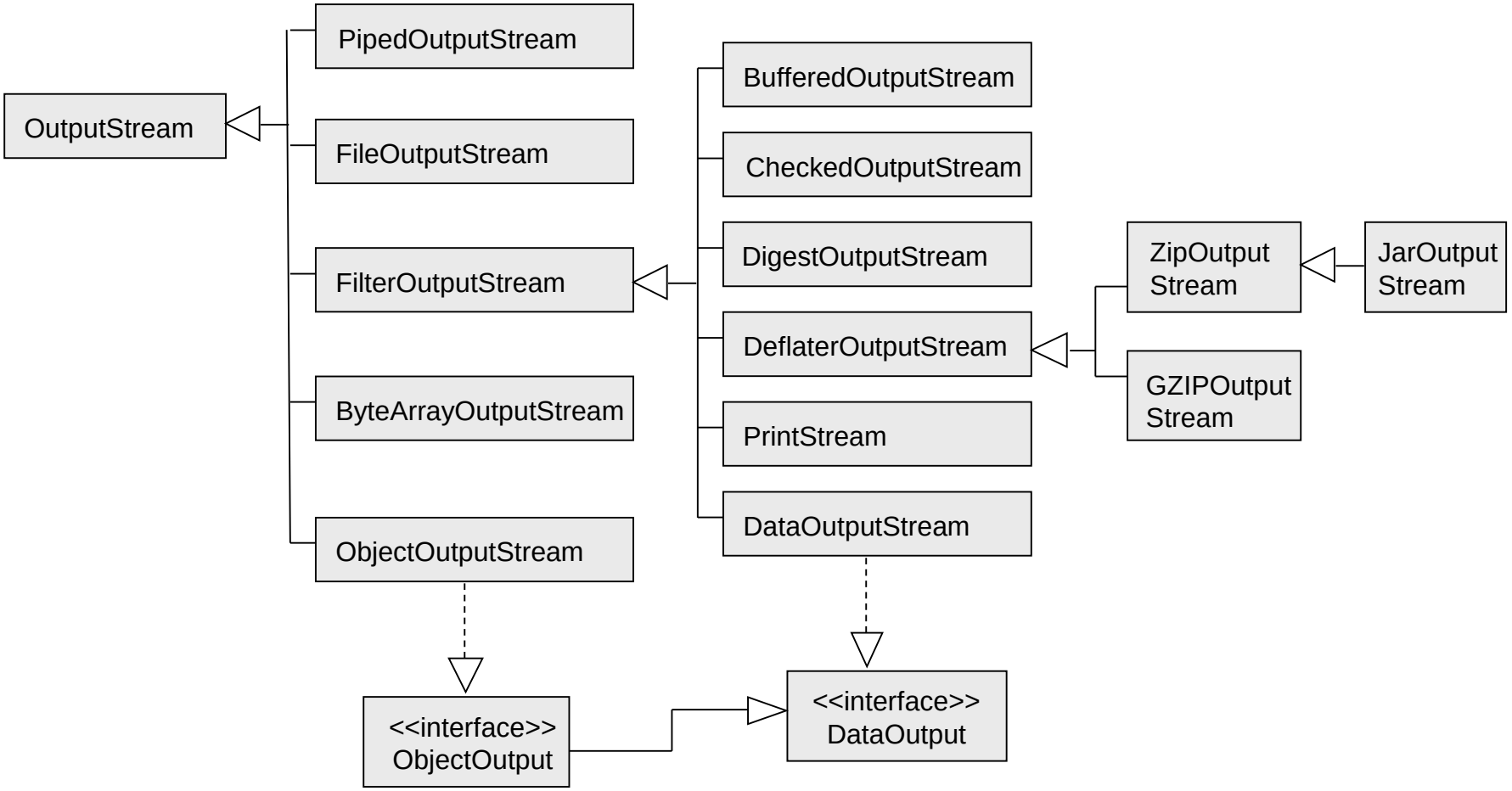
```
output:
0x40 0x09 0x21 0xfb  ;  @.!û
0x54 0x44 0x2d 0x18  ;  TD-.
```

InputStream Class Hierarchy



OutputStream Class Hierarchy

Excerpt!



Miscellaneous Stream Classes

```
public class PipedOutputStream
                    extends OutputStream {
  void connect(PipedInputStream snk);
  ...
}
```

```
public class PipedInputStream
                    extends InputStream {
  void connect(PipedOutputStream src);
  ...
}
```

- Pipes from output to input

```
public class PushbackInputStream
                    extends InputStream {
  int  read();
  void unread(int b);
  ...
}
```

- supports push back of date (`unread`)

```
public class PrintStream
                    extends FilterOutputStream {
  public void print(boolean b)
  public void print(int b)
  …
  public void println(boolean b)
  …
}
```

- formatted output
- by `print` und `println` methods

Standard Streams

Streams of `System`

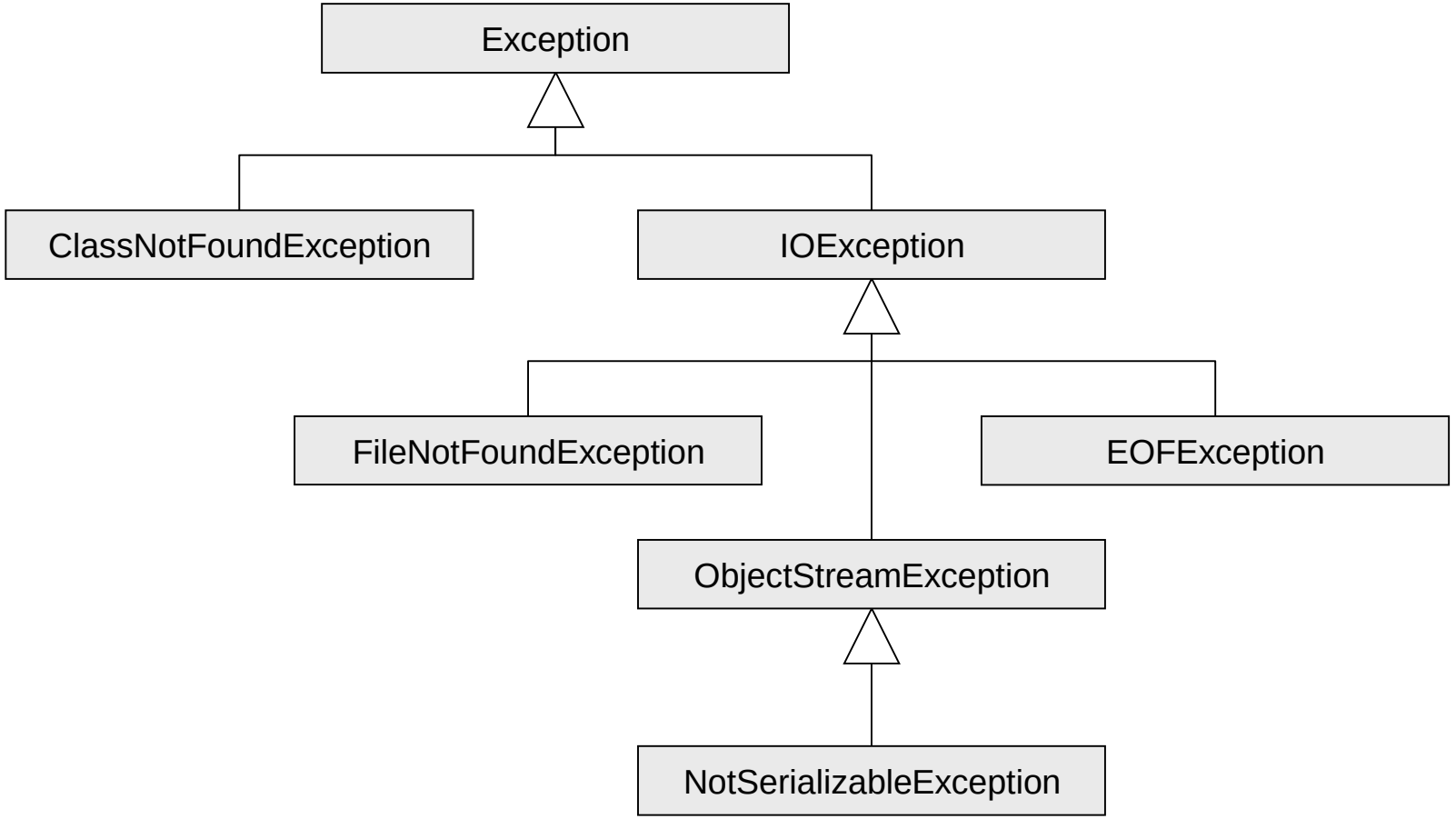
```
static InputStream in;   // the standard input stream
static PrintStream out;  // the standard output stream
static PrintStream err;  // the standard error stream
```

rerouting of standard streams

```
java Foo < input.txt > output.txt 2> error.txt
                              oder
java Foo > output.txt 2>&1
                              oder
java Foo >> output.txt
```

Hierarchy of Exceptions

Excerpt!



Streaming

Streams

Readers and Writers

Object serialization

Files

Summary

Character Input and Output

Input and output of character data by Readers and Writers

Base classes

- `Reader` (abstract base class analogous to `InputStream`)
- `Writer` (abstract base class analogous to `OutputStream`)

Data conversion

- Conversion of byte data to character data required
- `FileReader` und `FileWriter` use standard encoding (UTF8)
- other encodings by `InputStreamReader` and `OutputStreamWriter` or Character Sets

```
Writer out = new OutputStreamWriter(
                  new FileOutputStream("fname"),
                  "UTF8"         // encoding parameter
               );
```

Base Class Reader

Abstract method

```
read(char[] cbuf, int off, int len)
```

which fills char array

Other methods are based on method read

```
public abstract class Reader {
    public int read() throws IOException
    public int read(char[] cbuf) throws IOException
    public abstract int read(char[] cbuf, int off, int len)
                    throws IOException
    public long skip(long n) throws IOException
    public boolean ready() throws IOException
    public synchronized void mark(int readlimit)
    public synchronized void reset() throws IOException
    public boolean markSupported()
    public abstract void close() throws IOException
}
```

Base Class Writer

Abstract method

```
write(char[] cbuf, int off, int len)
```

which writes from char array

Other methods are based on method write

```
public abstract class Writer {
   public void write(int c) throws IOException
    public void write(char[] cbuf) throws IOException
    public abstract void write(char[] cbuf, int off, int len)
                    throws IOException
    public void write(String str) throws IOException
    public void write(String str, int off, int len)
                    throws IOException
    public abstract void flush() throws IOException
    public abstract void close() throws IOException
}
```

BufferedReader - PrintWriter

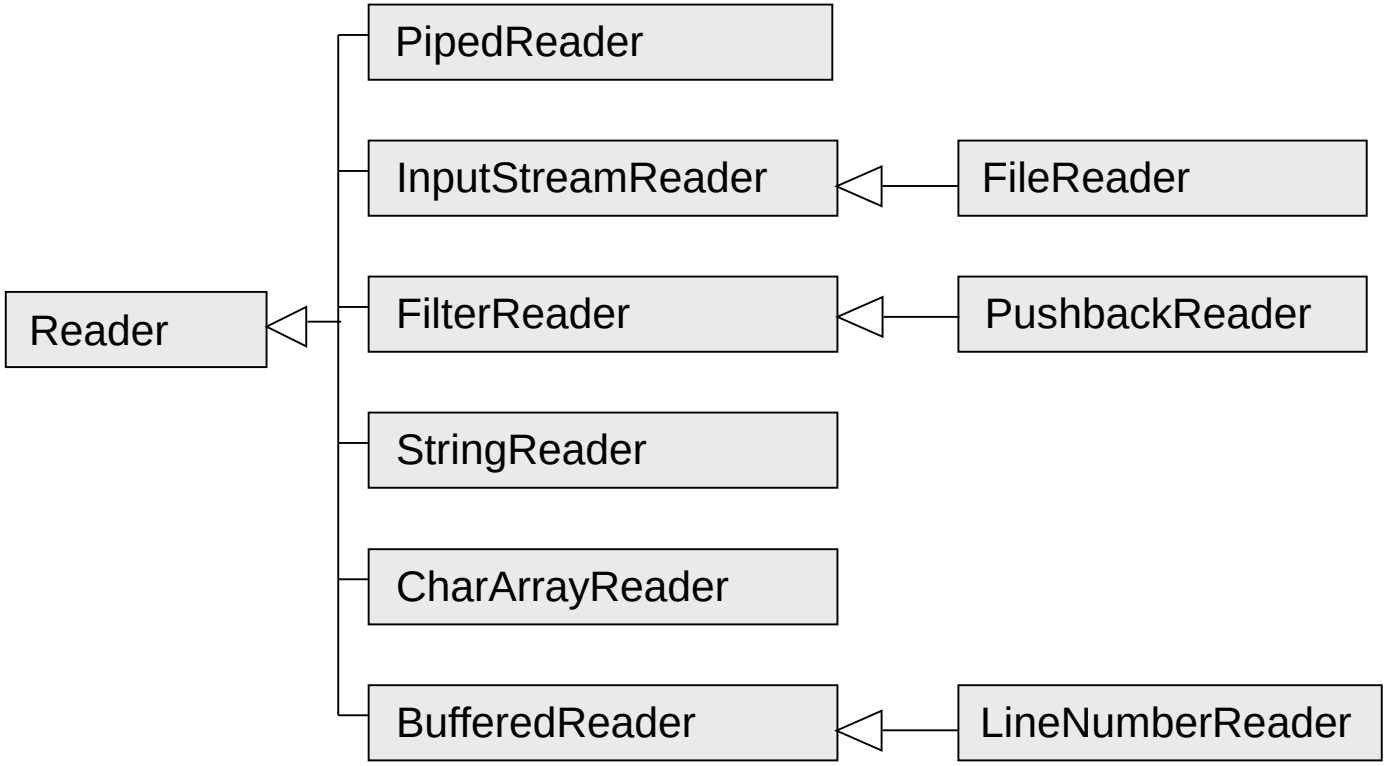
Line-based reading with `readLine`

Writing with `print/println` methods

```
try(
     BufferedReader reader = new BufferedReader(new InputStreamReader(instream));
     PrintWriter writer = new PrintWriter(new OutputStreamWriter(outstream));
    ) {
    String line = reader.readLine();
    while (line != null) {
        writer.println(line);
        line = reader.readLine();
    }
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

Reader Class Hierarchy

Excerpt!



Miscellaneous Readers

```
public class InputStreamReader extends Reader{
  public InputStreamReader(InputStream in)
  ...
}
```

- Bridge to input streams

```
public class LineNumberReader extends Reader {
  int  getLineNumber();
...
}
```

- access to line numbers

```
public class BufferedReader extends Reader {
  public String readLine() throws IOException
  ...
}
```

- buffered input
- line based reading (`readLine`)

```
public class PushbackReader extends Reader {
  public void unread(char[] cbuf,
         int off, int len) throws IOException
  ...
}
```

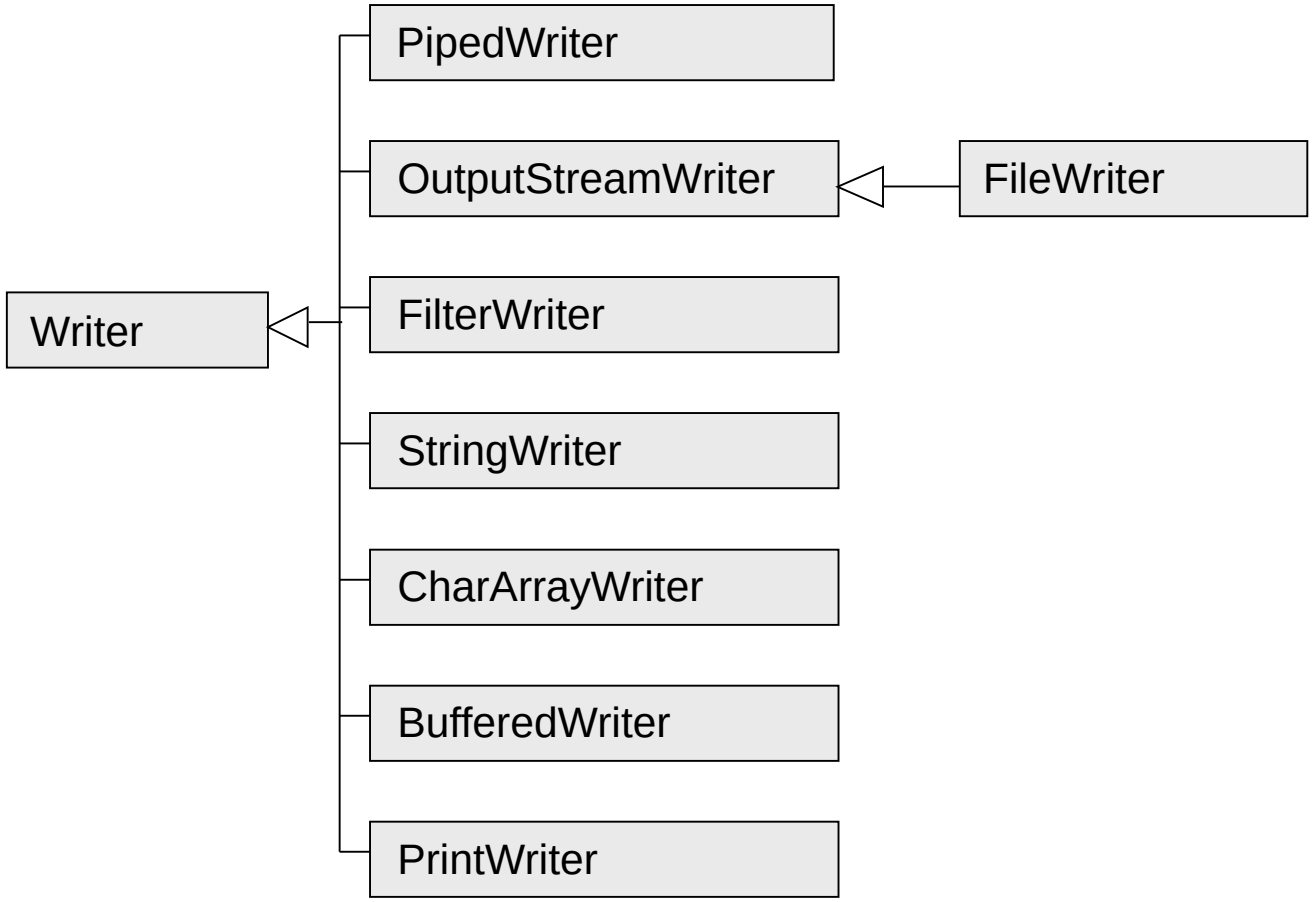
- supports pushback(`unread`)

```
public class PipedReader extends Reader {
  public void connect(PipedWriter src)
         throws IOException
  …
}
```

- piping output to input
- together with `PipedWriter`

Writer Class Hierarchy

Excerpt!



Miscellaneous Writers

```
public class OutputStreamWriter extends Writer

  public OutputStreamWriter  (OutputStream out)
  ...
}
```

- Bridge to `OutputStream`

```
public class PrintWriter extends Writer {
  public void print(boolean b)
  public void print(int b)
  …
  public void println(boolean b)
  …
}
```

- foratted text output
- with `print` und `println` methods

```
public class BufferedWriter extends Writer {
  public void newLine() throws IOException
  public void flush() throws IOException
  ...
}
```

- buffered output

```
public class PipedWriter extends Writer {
  public void connect(PipedReader snk)
         throws IOException
  …
}
```

- piping of output to input
- together with `PipedReader`

Streaming

Streams

Readers and Writers

Object serialization

Files

Summary

Object Serialization

Storing the state of objects

- Platform-independent transformation to bytes
- Used for storing objects and sending objects over a network

```
public class ObjectOutputStream {
  void writeBoolean(boolean data);
  ...
  void writeObject(Object obj);
}
```

```
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("data"));
Person p = ...;
out.writeObject(p);
```

Serializable

Class has to implement Serializable

- only objects implementing interface Serializable can be serialized

Fields get serialized automatically

- Fields of base data types
- Fields with types which are serializable

Transitive closure

- Objects which are stored in fields which are serializable are also serialized
- results in an object graph

Fields marked transient do not get serialized

- for avoiding serialization, e.g., when type not serializable

```
public interface Serializable { }
```

```
public class Person implements Serializable {
    private String name;
    private int age;
    private List<Person> children;
    private transient List<PropertyChangeListener> listener;
}
```

Serializable / Not serializable Data Types

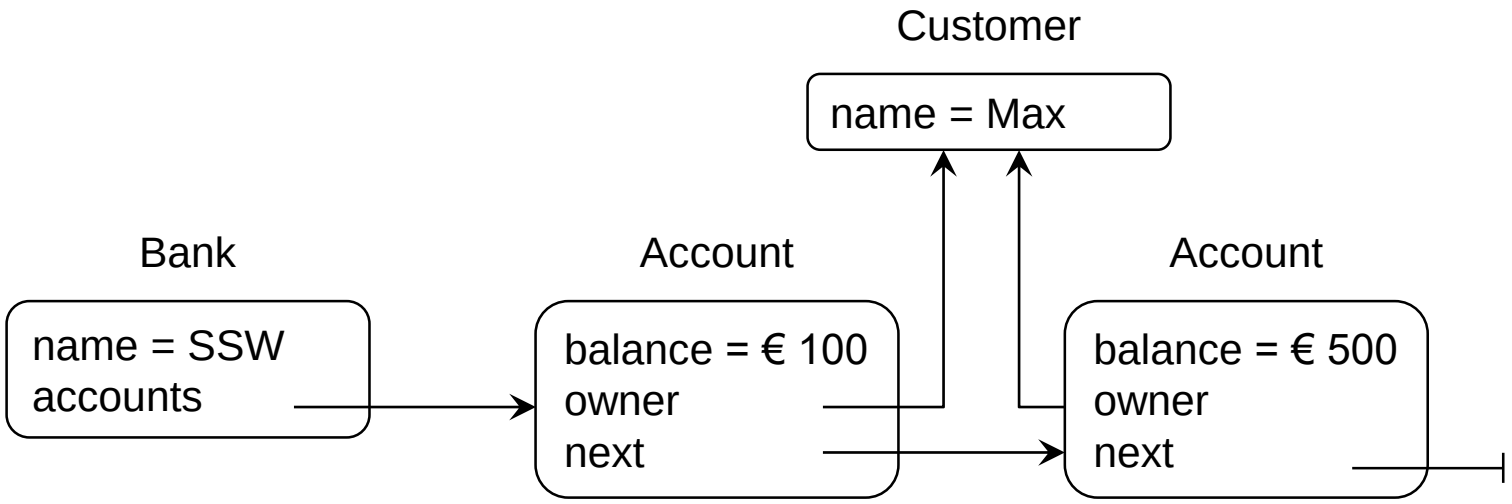
Serializable :

- built-in data types (`int`, `boolean`, `double`, …)
- arrays with serializable element type
- `String`
- Collections: `ArrayList`, `LinkedList`, `TreeSet`, …
- `Calendar`, `Date`, etc.
- …

Example of not serializable types :

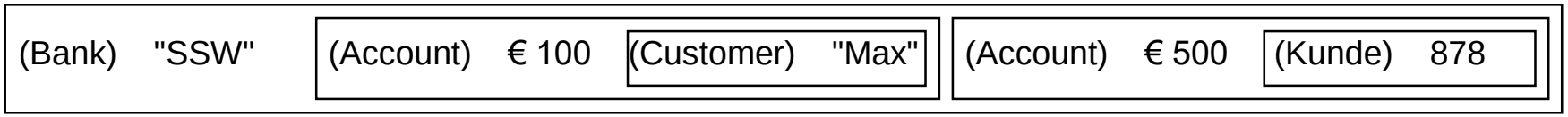
- `Thread`, `ClassLoader`, etc. ➔ Objects local to the VM
- `InputStream`, `Socket`, etc. ➔ Objects accessing os resources
- …

Serialization of Object Graph



Multiple reference to same object:

- 1st visit ➔ object is written completely
- other visits ➔ only ID of object is written



Deserialization

Reading object

- Allocation of memory
- Reading object data and setting fields
- Transient fields get standard value
- `ClassNotFoundException` if class not known

```
public class ObjectInputStream {
  boolean readBoolean();
  byte    readByte();
  ...
  Object  readObject();
  ...
}
```

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("data"));
Person p = (Person) in.readObject();
```

Serialization Format

Serialization contains

- type information
- version number
- data

```
public class Person implements Serializable {
    private String name;
    private int age;
}
```

```
Person hans = new Person("Hans", 26);
```

```
AC ED  00 05
73
72 00 06  Person
EB A3 AE 56 EF 40 EE 9B  02
00 02
L  00 04  name
74 00 12 Ljava/lang/String
I  00 03  age
78
70
74 00 04 Hans
00 00 00 00 00 1A
```

file indicator and version of serialization
73 for new object
72 for class, length of class name and class name
Serial UID, flag for implements Serializable
number of fields
L for field type class, length of field name, field name
74 for class type, length of type name, type name
I for type int, length of field name, field name
78 End of field description
70 no super class
data of field name
3 byte data for field age

User-defined Serialization (1/3)

Adapting the serialization by implementing methods `writeObject` and `readObject`

```
private void writeObject(ObjectOutputStream stream) throws IOException;
private void readObject(ObjectInputStream stream)
                            throws IOException, ClassNotFoundException;
```

```
class LabeledPoint implements Serializable {
    private String label;
    private transient Point2D.Double point;
    ...
    private void writeObject(ObjectOutputStream out)
            throws IOException {
        out.defaultWriteObject();
        out.writeDouble(point.x);
        out.writeDouble(point.y);
    }
    private void readObject(ObjectInputStream in)
            throws IOException, ClassNotFoundException {
        in.defaultReadObject();
        double x = in.readDouble();
        double y = in.readDouble();
        point = new Point2D.Double(x, y);
    }
}
```

Point2D not serializable
➔ **transient**

Default serialization of other fields incl. super class

serialization of data of transient field

Default deserialization

deserialization of data of transient field

Reconstruction of transient field

User-defined Serialization (2/3)

Special serialization by implementing interface `Externalizable`

```
public interface Externalizable extends java.io.Serializable {
  void writeExternal(ObjectOutput out) throws IOException;
  void readExternal(ObjectInput in) throws IOException,ClassNotFoundException;
}
```

- Serialization has to be implemented completely
- Only identity of class is automatically serialized
- Uses and requires standard constructor (without parameter)

```
class LabeledPoint implements Externalizable {
    ...
    public LabeledPoint() { }

    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeUTF(label);
        out.writeDouble(point.x);
        out.writeDouble(point.y);
    }

    public void readExternal(ObjectInput in)
                throws IOException, ClassNotFoundException {
        label = in.readUTF();
        double x = in.readDouble();
        double y = in.readDouble();
        point = new Point2D.Double(x, y);
    }
}
```

standard constructor

Writing data in writeExternal

Reading data and setting fields in readExternal

User-defined Serialization (3/3)

Replacing object by other object in serialization : `writeReplace` und `readResolv`

```
ANY-ACCESS-MODIFIER Object writeReplace() throws ObjectStreamException;
ANY-ACCESS-MODIFIER Object readResolve() throws ObjectStreamException;
```

- `writeReplace` will be called in writing object
- `readResolve` will be called when reaing object

Example: Reading object and replacing by singleton

```
class Orientation implements Serializable {
    public static final Orientation HORIZONTAL = new Orientation(1);
    public static final Orientation VERTICAL = new Orientation(2);

    private int value;

    private Orientation(int value) {
        this.value = value;
    }

    protected Object readResolve() throws ObjectStreamException {
        if (value == 1) return Orientation.HORIZONTAL;
        if (value == 2) return Orientation.VERTICAL;
        return null;
    }
}
```

After deserialization of object the singleton object is returned

Note: Serialization of enum values is done automatically !!

Version Control (1/2)

Problem: Compatibility of serialized object with new version of class

- in serialization process class gets unique ID (UID)

  ➔Hash-Code (SHA) of class definition

- When class changes class will be identified as incompatible

  ➔ Object cannot be read!

- However, often the class is compatible with old version,
  e.g., because only methods have changed

Version Control (2/2)

Solution:

- Setting `serialVersionUID` in source program
- the serialization UID will not be computed
- but the number from source code is taken ➔ programmer can define compatibility

Approach:

- Compute UID of old version of class by tool **`serialver`**

```
C:\>serialver Person
Person:    static final long serialVersionUID = 213554356225514906L;
```

- Setting UID as `serialVersionUID` in source code for new verions of class

```
public class Person implements Serializable {

   private static final long serialVersionUID = 213554356225514906L;
   …
```

Then new version compatible to old version

Streaming

Streams

Readers and Writers

Object serialization

Files

Summary

Class File

General representation of files and directories (both!!)

- Note: Creating File objects does not access file system
- Methods then will access file system, e.g. **`exists()`**

```
public class File {
  boolean  canRead();
  boolean  canWrite();
  boolean  delete();
  boolean  exists();
  String   getName()
  boolean  isDirectory();
  boolean  isFile();
  long     lastModified();
  long     length();
  String[] list();
  File[]   listFiles();
  ...
}
```

```
static long totalSize(File dir) {
  if (!dir.isDirectory()) {
    return -1;
  }
  long size = 0;
  File[] files = dir.listFiles();
  for (int i = 0; i < files.length; i++) {
    size += files[i].length();
  }
  return size;
}
```

Streaming

Streams

Readers and Writers

Object serialization

Files

Summary

SUMMARY

Streams

- connect to data source and sinks

Byte-oriented streams

- InputStream and OutputStream
- FileInputStream and FileOutputStream

Char-oriented streams

- Reader and Writer

Buffered streams

- BufferedInputStream and BufferedOutputStream
- BufferedReader and BufferedWriter

Serialization

- platform independent encoding of object graphs

➔ Remark: NIO provides new powerful concepts for file management and IO