Objektorientierte Programmiertechnik

Kapitel 12 – Ein-/Ausgabe und Streams

Übersicht

Inhalte

✓ Streams

Ein- und Ausgabe

Random Access Dateien

Serialisieren von Objekten

Das Stream Konzept

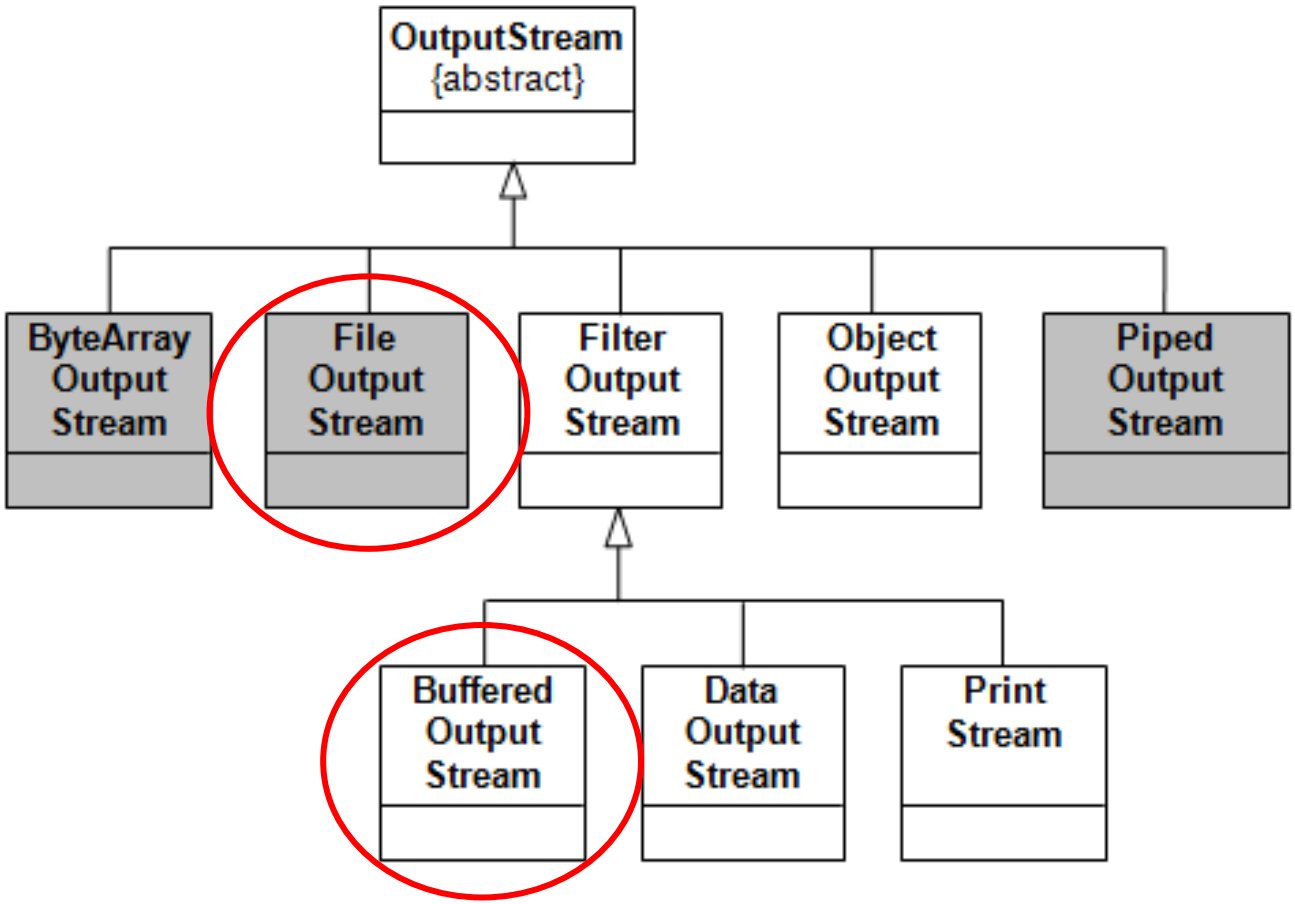
- Java verwendet das so genannte Stream-Konzept, um die komplizierten Einzelheiten der Kommunikation bei der Ein- und Ausgabe zu verbergen.
  - Ein Stream ist eine geordnete Folge von Bytes → Bytestrom

- Im Paket `java.io` befinden sich die entsprechenden Stream-Klassen
  - Byte-Streams
    - Lesender Zugriff: InputStreams
    - Schreibender Zugriff: OutputStreams
  - Character-Streams
    - Lesender Zugriff: Reader
    - Schreibender Zugriff: Writer

Basisklasse für	Bytes (oder Byte-Arrays)	Zeichen (oder Zeichen-Arrays)
Eingabe	`InputStream`	`Reader`
Ausgabe	`OutputStream`	`Writer`

OutputStream I

- Byte-OutputStream-Klassen



OutputStream II

BufferedOutputStream & FileOutputStream

- die in den Ausgabestrom geleiteten Bytes werden gepuffert
- ist immer nützlich, wenn viele Schreiboperationen gemacht werden (schnellere Verarbeitung)

```
…
public static void main(String[] args) throws IOException {

    FileOutputStream fos = new FileOutputStream("Bytes.txt");
    BufferedOutputStream bos = new BufferedOutputStream(fos);

    bos.write("Hallo Welt!".getBytes());
    bos.write("\n".getBytes());

    bos.flush();
    bos.close();
}
…
```

OutputStream III

BufferedOutputStream & FileOutputStream

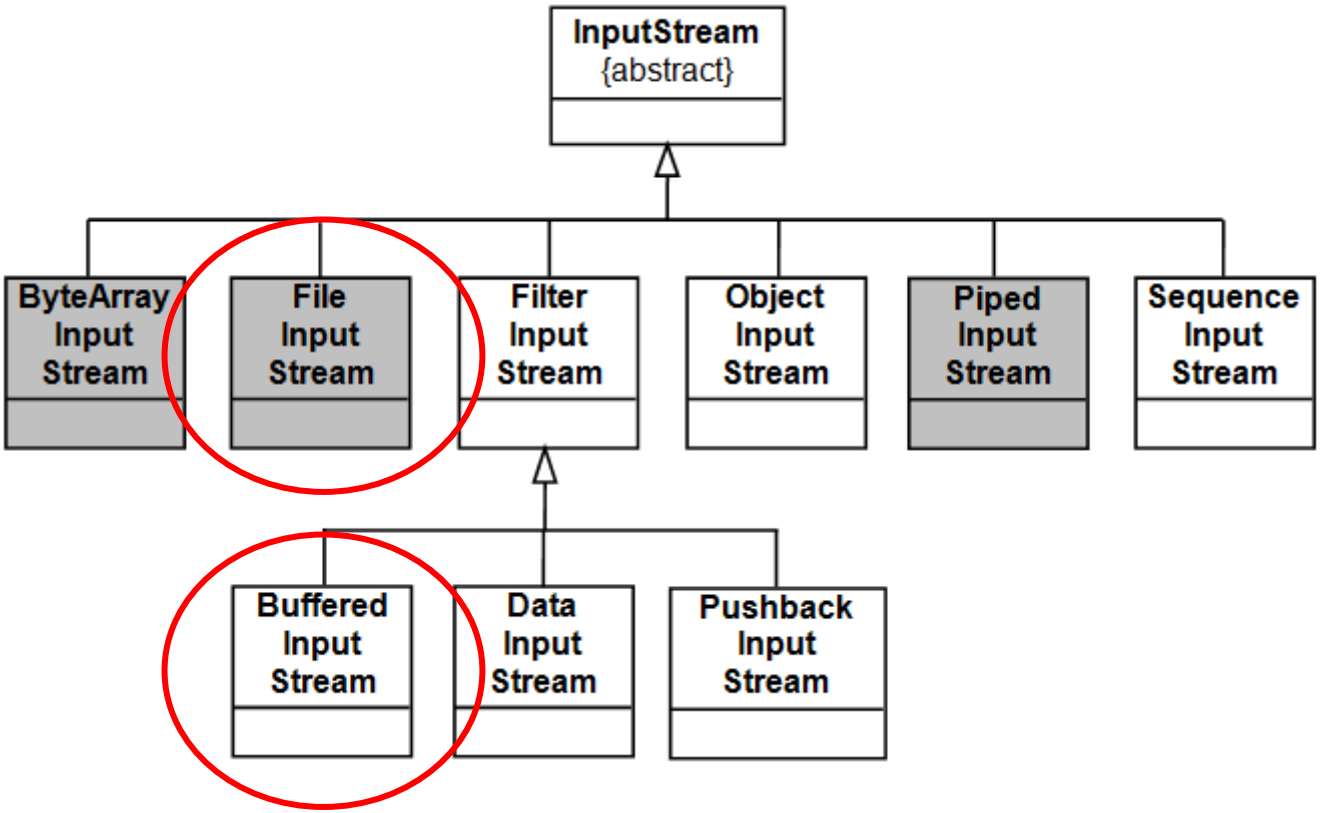
- „Echtes“ Beispiel für den praktischen Einsatz

```
//Construct the BufferedOutputStream object
BufferedOutputStream bufferedOutput;
try {
    bufferedOutput = new BufferedOutputStream(new FileOutputStream("X.txt"));

   bufferedOutput.write("Hallo Welt!".getBytes());

} catch (FileNotFoundException ex) {
   ex.printStackTrace();
} catch (IOException ex) {
   ex.printStackTrace();
} finally {
   //Close the BufferedOutputStream
   try {
        if (bufferedOutput != null) {
            bufferedOutput.flush();
            bufferedOutput.close();
        }
   } catch (IOException ex) {
            ex.printStackTrace();
   }
}
```

Demo in Eclipse:
OutputStreamDemo.java

InputStream I

- Byte-InputStream-Klassen



InputStream II

- BufferedInputStream & FileInputStream
  - die Eingabe funktioniert analog zur Ausgabe

```
…
public static void main(String args[]) throws IOException {

    FileInputStream fis = new FileInputStream("myFile.txt");

    BufferedInputStream bis = new BufferedInputStream(fis);

    int i;
    while ((i = bis.read()) != -1) {
      System.out.write(i);
    }

    bis.close();
    fis.close();
  }
…
```

InputStream III

- BufferedInputStream & FileInputStream
  - „Echtes“ Beispiel für den praktischen Einsatz

```
 FileInputStream fis = null;
    BufferedInputStream bis = null;
try {

    fis = new FileInputStream("myFile.txt");
    bis = new BufferedInputStream(fis);

    int i;
    while ((i = bis.read()) != -1) {
        System.out.write(i);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    // close the BufferedInputStream
    try {
        if(bis!=null)
            bis.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Demo in Eclipse:
InputStreamDemo.java

Übersicht

Inhalte

- ✓ Streams
- ✓ Ein- und Ausgabe
- Random Access Dateien
- Serialisieren von Objekten

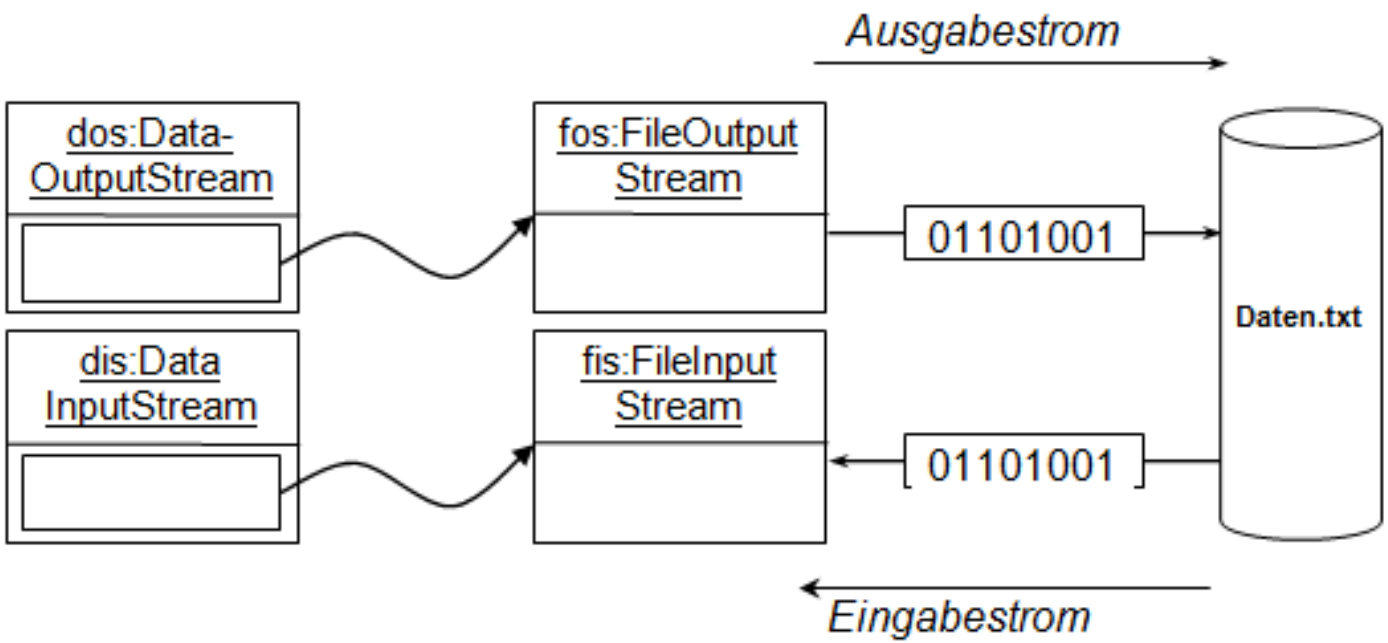
Elementare Datentypen I

- Datei-Ein- und –Ausgabe elementarer Datentypen
  - die beiden Klassen DataOutputStream und DataInputStream werden zur Ein- und Ausgabe elementarer Datentypen verwendet

```
import java.io.*;

public class EinUndAusgabeVonDatenPrimitiverTypen{

   public static void main (String[] args) throws IOException{

      FileOutputStream fos = new FileOutputStream ("Daten.txt");
      DataOutputStream dos = new DataOutputStream (fos);

      dos.writeInt (1);
      dos.writeDouble (1.1);
      dos.close();

      FileInputStream fis = new FileInputStream ("Daten.txt");
      DataInputStream dis = new DataInputStream (fis);

      System.out.println (dis.readInt());
      System.out.println (dis.readDouble());
      dis.close();
   }
}
```

Elementare Datentypen II

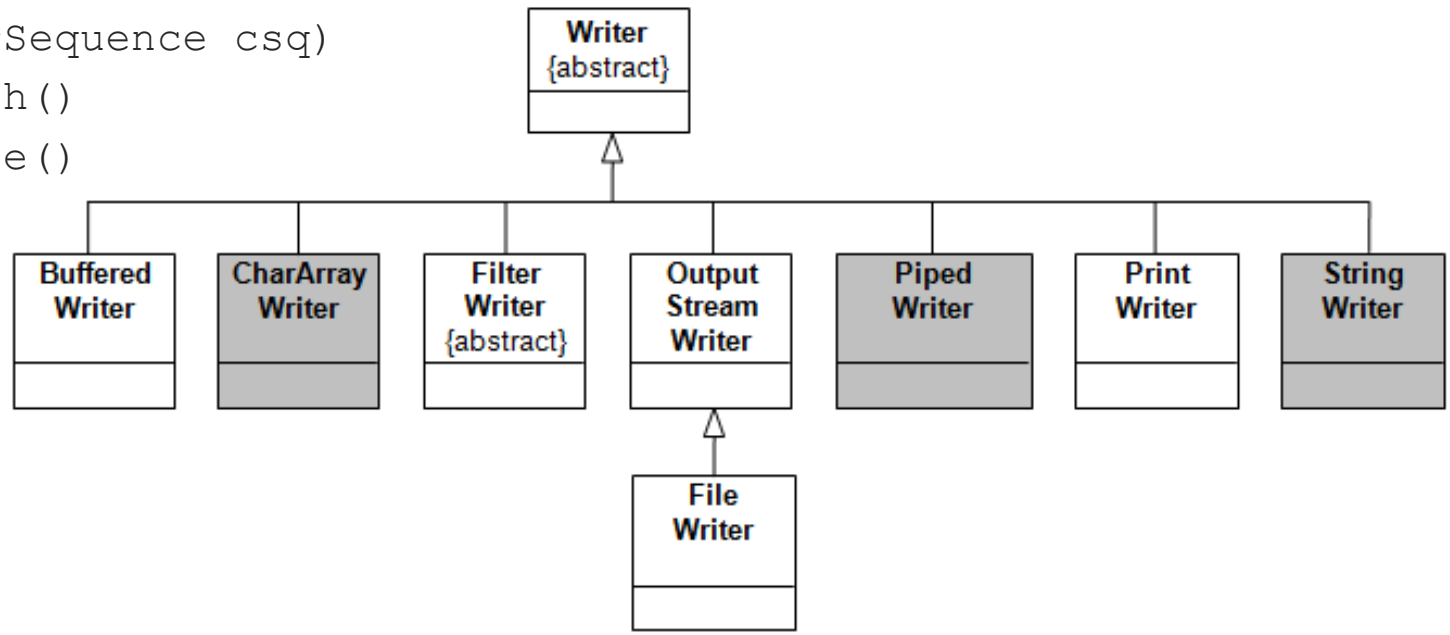
- Datei-Ein- und –Ausgabe elementarer Datentypen
  - die generierte Textdatei Daten.txt enthält die geschriebenen Informationen als Bytes



Characterstream - Klassen I

Characterstream – Klassen arbeiten mit Zeichen

- Die abstracte Basisklasse Writer deklariert folgende Methoden:
  - `void write(int c)`
  - `void write(char[] cbuf)`
  - `abstract void write(char[] cbuf, int off, int len)`
  - `void write(String str)`
  - `void write(String str, int off, int len)`
  - `Writer append(char c)`
  - `Writer append(CharSequence csq)`
  - `abstract void flush()`
  - `abstract void close()`



Characterstream - Klassen II

`BufferedWriter`

- Diese Klasse puffert Stream-Ausgaben von `write`.
  - Erst wenn der Puffer voll ist, oder die Methode `flush` aufgerufen wird, werden die gepufferten Daten in den Stream geschrieben.
  - Wenn in eine Datei geschrieben wird, reduziert sich so die Anzahl der Zugriffe und die Performance wird erhöht.

- `BufferedWriter` besitzt zwei Konstruktoren:
  - `public BufferedWriter(Writer out)`
  - `public BufferedWriter(Writer out, int size)`

- In beiden Fällen wird ein bereits existierender `Writer` übergeben.
- Die `write`-Methoden entsprechen denen der Klasse `Writer`.
- Zusätzliche Methode: `newLine`

Characterstream - Klassen III

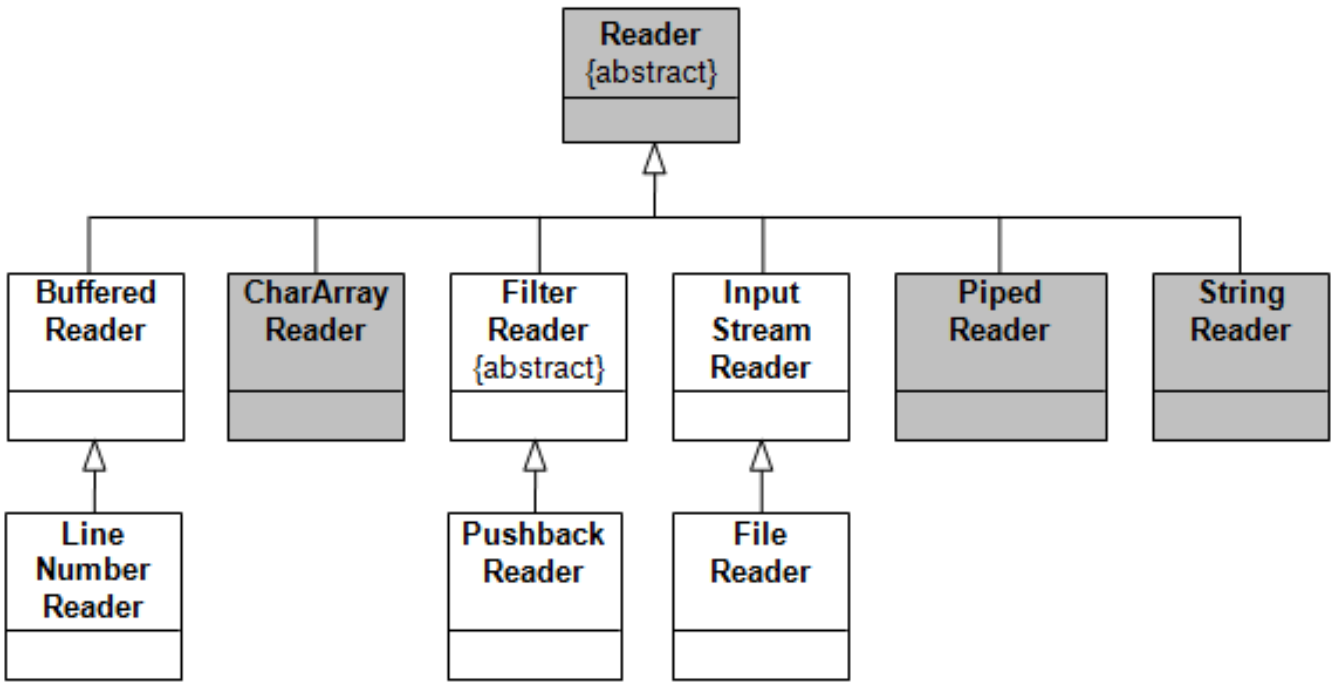
- BufferedWriter

```
import java.io.*;
public class BufferedWriterDemo {
    public static void main(String[] args) {

        Writer f1;
       BufferedWriter f2;
       String s;
       try{
           f1 = new FileWriter("buffer.txt");
           f2 = new BufferedWriter(f1);
           //f2 = new BufferedWriter(new FileWriter("buffer.txt"));
           for(int i=1; i<=10000; i++){
               s = "Dies ist die " + i + ". Zeile.";
               f2.write(s);
               f2.newLine();
           }
           f2.flush();
           f2.close();
           f1.close();
       }catch(IOException e){
           System.err.println("Fehler beim Erstellen der Datei!");
       }
    }
}
```

Siehe Demo in Eclipse:
BufferedWriterDemo.java

Characterstream - Klassen IV

- Basisklasse `reader`
  - Diese abstracte Basisklasse deklariert folgende Methoden:
    - `abstract int read(char[] cbuf, int off, int len)`
    - `int read(CharBuffer target)`
    - `int read()`
    - `int read(char[] cbuf)`
    - `boolean ready()`
    - `abstract void close`



Characterstream - Klassen V

`BufferedReader`

- Diese Klasse dient der Pufferung von Eingaben und kann verwendet werden, um die Performance beim Lesen von externen Dateien zu erhöhen.

- `BufferedReader` besitzt zwei Konstruktoren:
  - `public BufferedReader(Reader in)`
  - `public BufferedReader(Reader in, int size)`

  - In beiden Fällen wird ein bereits existierender Reader übergeben.
  - Zusätzliche Methode: `readLine` → liest eine komplette Textzeile ein und gibt diese als String an Aufrufer zurück (ohne Begrenzungszeichen \r\n)

Characterstream - Klassen VI

- BufferedReader

```
import java.io.*;

public class BufferedReaderDemo {

    public static void main(String[] args) {

        BufferedReader f;
        String line;
        try{
            f = new BufferedReader(new FileReader("buffer.txt"));

            while((line = f.readLine()) != null){
                System.out.println(line);
            }

            f.close();

        }catch(IOException e){
            System.err.println("Fehler beim Lesen der Datei!");
        }
    }
}
```

Siehe Demo in Eclipse:
BufferedReaderDemo.java

Aufgabe 12.01

Aufgabe 12.01 Einmaleins

a) Schreiben Sie ein Programm, das die Zahlen des kleinen Einmaleins berechnet und in der Textdatei `einmaleins.txt` ablegt. Die Zahlen sollen jeweils durch einen Tabulator voneinander getrennt werden. Die Textdatei einmaleins.txt sollte folgenden Inhalt haben:

```
1     2     3     4     5     6     7     8     9     10
2     4     6     8     10    12    14    16    18    20
3     6     9     12    15    18    21    24    27    30
4     8     12    16    20    24    28    32    36    40
5     10    15    20    25    30    35    40    45    50
6     12    18    24    30    36    42    48    54    60
7     14    21    28    35    42    49    56    63    70
8     16    24    32    40    48    56    64    72    80
9     18    27    36    45    54    63    72    81    90
10    20    30    40    50    60    70    80    90    100
```

`Hinweis:  new BufferedWriter(new FileWriter("einmaleins.txt"));`

b) Erweitern Sie das Programm so, dass die Größe der berechneten Tabelle durch den Benutzer angegeben werden kann. Die Größe soll als Parameter beim Programmaufruf übergeben werden, wie in folgendem Beispielaufruf: `java Einmaleins 100`

Aufgabe 12.01

Aufgabe 12.01 Einmaleins

c) Lesen Sie die Datei `einmaleins.txt` ein und bilden Sie für jede eingelesene Zeile die Summe der Tabulator-getrennten Zahlen, z.B. für Zeile 1:

```
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
…
```

Hinweis:

- `new BufferedReader(new FileReader("einmaleins.txt"));`
- bilden Sie ein String-Array aus jeder Zeile, wobei jede Zahl als ein Element gespeichert wird. Verwenden Sie dafür die String-Funktion `String[] split(String regex)` mit dem Tabulator ("\t") als Trennzeichen.
  - um die Summe zu bilden, müssen Sie jedes Element zu `int` parsen.

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. ~~Geschachtelte Klassen~~
12. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Streams
- ✓ Ein- und Ausgabe
- ✓ Random Access Dateien

Serialisieren von Objekten

Random-Access-Dateien I

Dateien und Verzeichnisse mit der Klasse File

- Ein File-Objekt repräsentiert einen Datei- oder Verzeichnisnamen im Dateisystem.
- Um ein File-Objekt zu erzeugen, kann folgender Konstruktor verwendet werden:
- `File(String pathname)`

```
…
File f = new File("C:\\test.txt");
…
```

- Die Angabe des Pfades ist wegen der Pfadtrenner plattformabhängig.
- Verwenden Sie falls erforderlich das Konstrukt: `File.separator`

```
…
File f = new File("C:" + File.separator + "test.txt");
…
```

- Weitere Informationen zur File-Klasse entnehmen Sie bitte der API.

Aufgabe 12.02

Aufgabe 12.02 Verzeichnisse auslesen

Schreiben Sie eine Klasse `Aufg_13_02`, die in Ihrer `main`-Methode aus einem als Argument übergebenen Verzeichnispfad sämtliche Namen der beinhaltenden Dateien, auch aus Unterverzeichnissen, ausgibt.

Hinweis:

Implementieren Sie eine rekursive Methode, welche den Dateinamen des File-Objekts ausgibt bzw. welche sich selbst aufruft, wenn das aktuelle File-Objekt ein Verzeichnis ist.

Recherchieren Sie die notwendigen File-Methoden in der Java-API.

Random-Access-Dateien II

- Neben der Stream-basierten Ein- und Ausgabe bietet Java auch die Möglichkeit des wahlfreien Zugriffs auf Dateien.
  - Datei kann an beliebiger Stelle gelesen oder beschrieben werden.

- Es gibt für RandomAccessFile zwei Konstruktoren:
  - `public RandomAccessFile(File file, String mode)`
  - `public RandomAccessFile(String name, String mode)`

  - `mode: r (read) oder rw (read and write)`

Random-Access-Dateien III

- Positionierung des Dateizeigers
- `RandomAccessFile` stellt eine Reihe von Methoden zum Zugriff auf den Satzzeiger zur Verfügung:
  - `public long getFilePointer()`
    liefert die aktuelle Position des Satzzeigers
  - `public void seek(long pos)`
    positioniert den Satzzeiger an die Stelle pos (immer vom Anfang der Datei aus betrachtet)
  - `public  void skipBytes(int n)`
    Positionierung des Satzzeigers relativ zur aktuellen Position (n kann dabei auch negative Werte haben)
  - `public long length()`
    liefert die Länge der Datei in Bytes

Weitere Methoden: siehe Java-API

Random-Access-Dateien IV

Lesezugriffe

```
public final boolean readBoolean();
public final byte readByte();
public final char readChar();
public final double readDouble();
public final float readFloat();
public final int readInt();
public final long readLong();
public final short readShort();
public final String readUTF();
public final void readFully(byte b[]);
public final void readFully(byte b[], int off, int len);
public final String readLine();
public final int readUnsignedByte();
public final int readUnsignedShort();
```

Schreibzugriffe

```
public final void writeBoolean(boolean v);
public final void writeByte(int v);
public final void writeBytes(String s);
public final void writeChar(int v);
public final void writeChars(String s);
public final void writeDouble(double v);
public final void writeFloat(float v);
public final void writeInt(int v);
public final void writeLong(long v);
public final void writeShort(int v);
public final void writeUTF(String str);
```

Random-Access-Dateien V

- Beispiel

```
import java.io.*;
public class RandomAccessFileDemo {
    public static void main(String[] args) {
        try {
            RandomAccessFile f = new RandomAccessFile("buffer.txt", "rw");
            // lesen
            String s = f.readLine();
            System.out.println(s);
            f.seek(100);
            s = f.readLine();
            System.out.println(s);
            // schreiben
            f.seek(0);
            f.write("**************************\n".getBytes());
            f.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Siehe Demo in Eclipse:
RandomAccessFileDemo.java

Random-Access-Dateien Aufgabe

Random-Access-Dateien – Aufgabe

- Legen Sie eine Textdatei mit folgendem Inhalt an: 8 Zeilen mit 8 Nullen
- Öffnen Sie diese Textdatei als `RandomAccessFile`.
- Geben Sie den Dateiinhalt aus.
- Nun soll eine Diagonale aus Einsen in die Textdatei eingefügt werden.

  Also:

  Zeile 1; erste Null wird zur 1

  Zeile 2; zweite Null wird zur 1

  etc.

- Geben Sie den Dateiinhalt erneut aus.

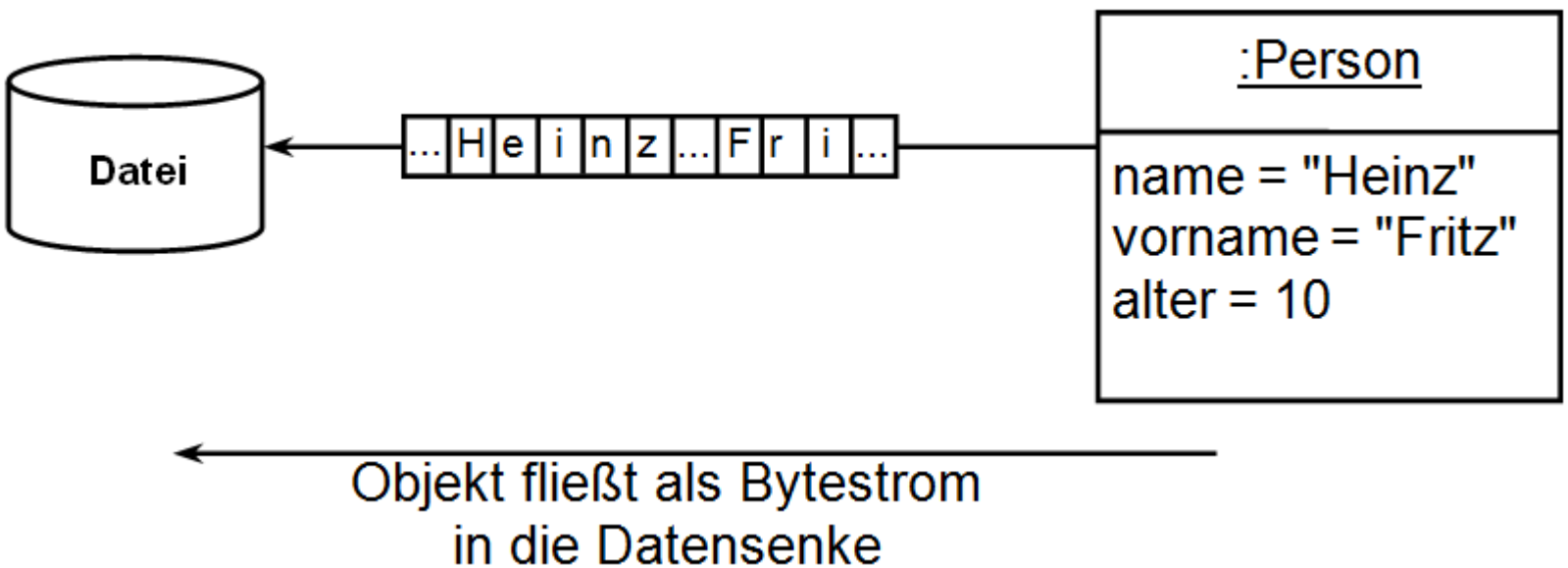
Übersicht

Inhalte

- ✓ Streams
- ✓ Ein- und Ausgabe
- ✓ Random Access Dateien
- ✓ Serialisieren von Objekten

Ein und Ausgabe von Objekten

- In Java können nicht nur Werte elementarer Datentypen gespeichert und gelesen werden, sondern auch ganze Objekte.
- Datenfelder eines Objektes werden in einen Bytestrom überführt und schließlich gespeichert.



Objektserialisierung I

- Die Klasse `ObjectOutputStream`
  - Klasse für das Überführen der Datenfelder eines Objektes in einen Bytestrom
- Die Klasse `ObjectInputStream`
  - Klasse für das Überführen eines Bytestroms in ein Objekt
- Die Schnittstelle `Serializable`
  - Dient als Kennzeichnung, dass ein Objekt einer Klasse serialisierbar ist.
  - Die Schnittstelle selbst deklariert keine Methoden.

```
class Person implements Serializable
{
   private String name;
   private String vorname;
   …
}
```

Objektserialisierung II

- Beispiel: Objekt schreiben

```
public class Serial
{
   public static void main (String[] args) throws Exception
   {
      // Datei text.txt zum Schreiben öffnen.
      ObjectOutputStream out =
         new ObjectOutputStream (new FileOutputStream ("obj.dat"));
      Person pers1 = new Person ("Weiss", "Renate", 12);
      Person pers2 = new Person ("Maier", "Anja", 13);

      // Objekte pers1 und pers2 in die Datei schreiben.
      out.writeObject (pers1);
      out.writeObject (pers2);
      out.close();
  }
}
```

Objektserialisierung III

- Beispiel: Objekt lesen

```
public class Serial
{
   public static void main (String[] args) throws Exception
   {
      // Datei text.txt zum Lesen öffnen.
      ObjectInputStream in =
         new ObjectInputStream (new FileInputStream („obj.dat"));

      // Die Methode readObject() gibt eine Referenz vom Typ
      // Object zurück. Es muss ein expliziter Cast erfolgen
      Person p1 = (Person) in.readObject();
      Person p2 = (Person) in.readObject();

      in.close();
  }
}
```

Siehe Demo in Eclipse:
Serial.java

Objektserialisierung Aufgabe

Aufgabe Serialisierung

a) Implementieren Sie eine serialisierbare Klasse Auto mit den Instanzvariablen `int ps` und `String hersteller` mit den zugehörigen `getter`-Methoden.

b) Implementieren Sie eine weitere Klasse `AutoTest`, in der Sie in der `main`-Methode ein Instanz von Auto erzeugen (über Konstruktor mit Parameterübergabe von `ps` und `hersteller`).

c) Serialisieren Sie diese Auto-Instanz.

d) Danach deserialisieren Sie das Objekt und geben Sie die Instanzvariablen `ps` und `hersteller` über die Systemausgabe aus.

Das Schlüsselwort transient

- Die Serialisierung von Instanzvariablen kann unterdrückt werden, in dem das Schlüsselwort `transient` bei der Deklaration der Instanzvariablen angegeben wird.
  - Z.B. Instanzvariablen alter oder passwort etc.

```
class Person implements Serializable
{
   private String name;
   private String vorname;
   private transient int alter;
   …
}
```

Die Schnittstelle Externalizable I

- Mit Hilfe der Schnittstelle `Serializable` kann ***KEIN*** Einfluss auf die Serialisierung genommen werden
  - z.B. Abänderung des Ausgabeformates

- Für solche Fälle existiert die Schnittstelle `Externalizable`
  - Ist von `Serializable` abgeleitet
  - Erweiterung um die beiden folgenden Methoden:

```
public void writeExternal(ObjectOutput out) throws IOException

public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException
```

Die Schnittstelle Externalizable II

- Beispiel

```
class Person2 implements Externalizable
{
   private String name;
   private String vorname;
   private int alter;
   …

   public void writeExternal(ObjectOutput out) throws IOException
   {
      System.out.println("Explizites Schreiben!");
      out.writeObject(name);
      out.writeObject(vorname);
      out.writeInt(alter);
   }
   public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException
   {
      System.out.println("Explizites Lesen!");
      name = (String) in.readObject();
      vorname = (String) in.readObject();
      alter = in.readInt();
   }
}
```

Die Schnittstelle Externalizable III

- Beispiel

```
public class Serial2
{
   public static void main (String[] args) throws Exception
   {
      ObjectOutputStream out =
         new ObjectOutputStream (new FileOutputStream ("obj.dat"));
      Person2 pers1 = new Person2 ("Mustermann", "Heinz", 45);
      Person2 pers2 = new Person2 ("Heinzelmann", "Max", 30);

      // Objekte pers1 und pers2 in die Datei schreiben.
      out.writeObject (pers1);
      out.writeObject (pers2);
      out.close();

      ObjectInputStream in =
         new ObjectInputStream (new FileInputStream ("obj.dat"));

      // Datentypen wieder einlesen
      ((Person2) in.readObject()).print();
      ((Person2) in.readObject()).print();
      in.close();
   }
}
```

Aufgabe 12.03

Aufgabe 12.03 BenutzerLogin Teil 1

Es soll die Klasse `BenutzerLogin` geschrieben werden. Diese Klasse kapselt einen Login mit den Attributen `name` vom Typ `String`, `passwort` vom Typ `String` und `online` vom Typ `boolean` und besitzt folgende Methoden:

- einen `Konstruktor`, der es ermöglicht, den Namen und das Passwort für den neu erzeugten Login zu setzen.
- `anmelden (String password)` zum Vergleichen der Passwörter und zum Setzen von online auf true, falls die Passwörter gleich sind.
- `print()` zur Ausgabe des Benutzernamens und der Information, ob der Benutzer online ist.

Aufgabe 12.03

Aufgabe 12.03 BenutzerLogin Teil 2

a) Implementieren Sie die Klassen `BenutzerLogin (implements Serializable)` und `TestLogin`.

b) Erzeugen Sie in der `TestLogin`-Klasse ein Objekt der Klasse `BenutzerLogin` und melden Sie sich bei diesem Objekt an. Überprüfen Sie hierbei, ob der Anmeldevorgang erfolgreich war, indem Sie die Methode `print()` vor und nach dem Anmelden aufrufen.

c) Speichern Sie das Objekt mit Hilfe der Objektserialisierung (`ObjectOutputStream`) in einer Datei. Beachten Sie hierbei, dass die Variable online nicht serialisiert (`transient`) werden soll.

d) Stellen Sie das Objekt mit Hilfe der Klasse `ObjectInputStream` wieder her und überprüfen Sie, ob der Benutzer immer noch angemeldet ist, indem Sie die Methode `print()` des neu erzeugten Objektes aufrufen.

e) Schreiben Sie das Programm um, damit statt `Serializable` die Schnittstelle `Externalizable` verwendet wird. Passen Sie das Programm der neuen Schnittstelle an und beachten Sie, dass weiterhin die bisherige Funktionalität erhalten bleiben soll.