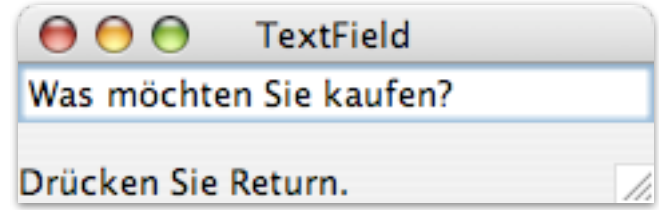
Rückblick

Wie funktioniert *Event-Handling* in Java?
Wozu dient ein *Listener-Interface*?

Unterschied zwischen *Hintergrundkomponenten* und *interaktiven* Komponenten?

Welche Aufgabe haben *Layoutmanager*?

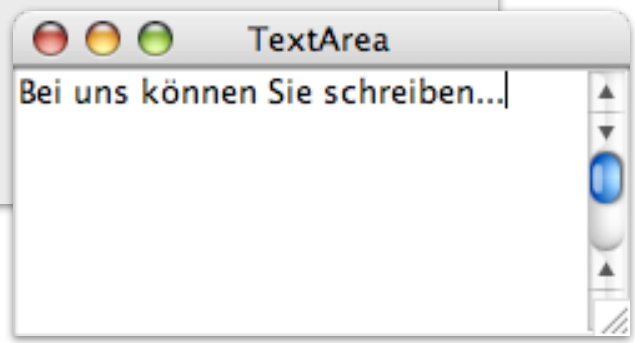
Swing-Komponenten: JTextField und JLabel




```
JTextField field = new JTextField(30);      // Textfeld mit 30 Spalten
JLabel label = new JLabel ("Drücken Sie Return.");

System.out.println (field.getText());       // Inhalt ausgeben
label.setText (field.getText());            // Label-Text setzen

field.setText("");                          // Inhalt löschen
field.setText("Was möchten Sie kaufen?");

field.selectAll();                          // Text im Feld auswählen
field.requestFocus();                       // Cursor in das Feld zurücksetzen

// für Enter/Return-ActionEvents registrieren (Interface ActionListener)
field.addActionListener(this);
```

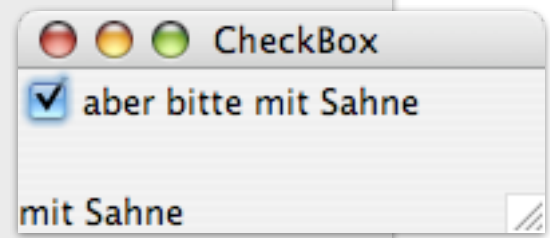
Swing-Komponenten: JTextArea

```
JTextArea text = new JTextArea(30, 50);     // 30 Zeilen, 50 Spalten
text.setLineWrap(true);                     // mit Zeilenumbruch

// JScrollPane für den Textbereich mit vertikalem Scrollbalken
JScrollPane scroller = new JScrollPane(text);
scroller.setVerticalScrollBarPolicy
(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
scroller.setHorizontalScrollBarPolicy
(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

text.setText("Den vorhandenen Text ersetzen");
text.append("oder Text anhängen");
text.seletcAll();
text.requestFocus();
```

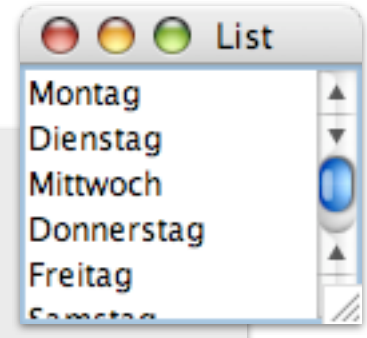


Swing-Komponenten: JCheckBox

```
JCheckBox check = new JCheckBox("aber bitte mit Sahne");

// Checkbox voreinstellen (true bzw. false)
check.setSelected(true);

// für ItemEvents registrieren, wenn Häckchen gesetzt/entfernt wird
check.addItemListener(this);

// Event behandeln (Interface ItemListener)
public void itemStateChanged(ItemEvent event) {
   if (check.isSelected()) {
      // Checkbox ist ausgewählt
   }
}
```



Swing-Komponenten: JList



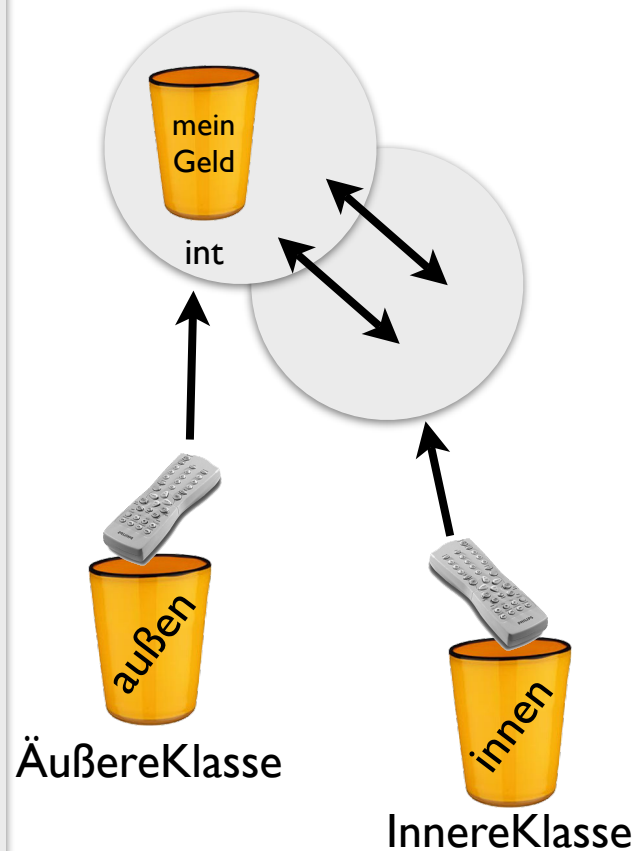

```
String[] entries = {"Motag","Dienstag", ...,"Sonntag"};
JList list = new JList(entries);

// Scrollbalken wie bei JTextArea hinzufügen
...
list.setVisibleRowCount(5);  // Anzahl der sichtbaren Zeilen

// nur einen Eintrag auf einmal auswählen
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

// für Listenauswahl-Events registrieren
list.addListSelectionListener(this);

// Event behandeln (Interface ListSelectionListener)
public void valueChanged(ListSelectionEvent event) {
    if (!event.getValueIsAdjusting()) {
        String selection = (String) list.getSelectedValue();    // liefert ein Object zurück
    }
}
```

Innere Klassen

```
class ÄußereKlasse {
    private int meinGeld;

    // die innere Klasse hat Zugriff auf alle
    //Variablen und Methoden der äußeren Klasse
    class InnereKlasse {
        void tuWas() {
            meinGeld = 1000000;
        }
    }
}
```

Innere Klassen instantiieren

```
class ÄußereKlasse {
    private int meinGeld;

    // Eine Instanz der inneren Klasse erzeugen
    InnereKlasse innen = new InnereKlasse();

    void machWas() {
        innen.tuWas();
    }

    class InnereKlasse {
        // die innere Klasse ist an die äußere Klasse gebunden
        void tuWas() {
            meinGeld = 1000000;
        }
    }
}
```

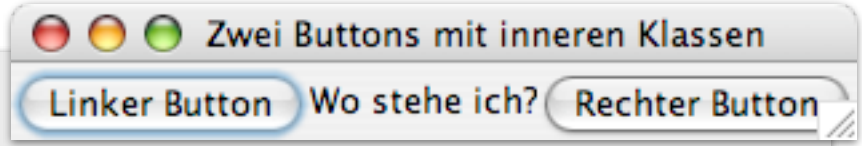


Innere Klassen

für

Action-Events

verschiedener Buttons.




```
class ZweiButtons {
    public void los() {
        // Fenster initialisieren, etc. (siehe EinfachesGUIEvent-Beispiel)
        ...
        JButton linksButton = new JButton("Linker Button");
        JButton rechtsButton = new JButton("Rechter Button");

        // jeder Button erhält eine Instanz seiner eigenen Listener-Klasse
        linksButton.addActionListener(new LinksButtonListener());
        rechtsButton.addActionListener(new RechtsButtonListener());
        ...
    }

    // jede innere Klasse ist für die Events eines Buttons zuständig
    class LinksButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            // linker Button wurde geklickt, hier darauf reagieren
        }
    }
    class RechtsButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            // rechter Button wurde geklickt, hier darauf reagieren
        }
    }
}
```

Jetzt sind Sie wieder an der Reihe!

Lesen Sie zu grafischen Benutzeroberflächen
Kapitel 12 und 13.

Serialisierung und Datei-E/A

Daten speichern

Textdateien verstehen alle Programme:
Daten/Werte beim Speichern durch Feldtrennzeichen trennen, z.B. durch Leerzeichen, Komma, Semikolon...

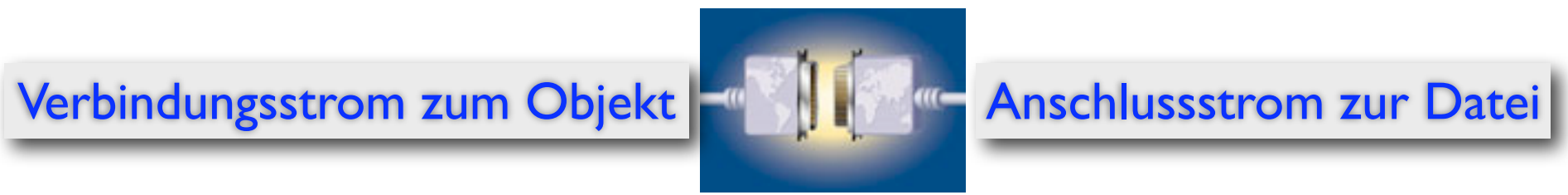
Serialisierung verstehen nur Java-Programme:
Objekte beim Speichern "flach drücken" und beim Lesen "aufblasen" (Interface Serializable).

serialisiert   **deserialisiert**

Anschluss- und Verkettungsströme

Anschlussströme repräsentieren die Verbindung zu Dateien, zum Zielort oder zur Quelle, und bieten "low-level" Methoden zum Schreiben/Lesen von Bytes.

Verkettungsströme bieten "high-level" Methoden zum Serialisieren/Deserialisieren von Objekten. Sie müssen mit anderen Strömen verkettet werden.



Objekte serialisieren

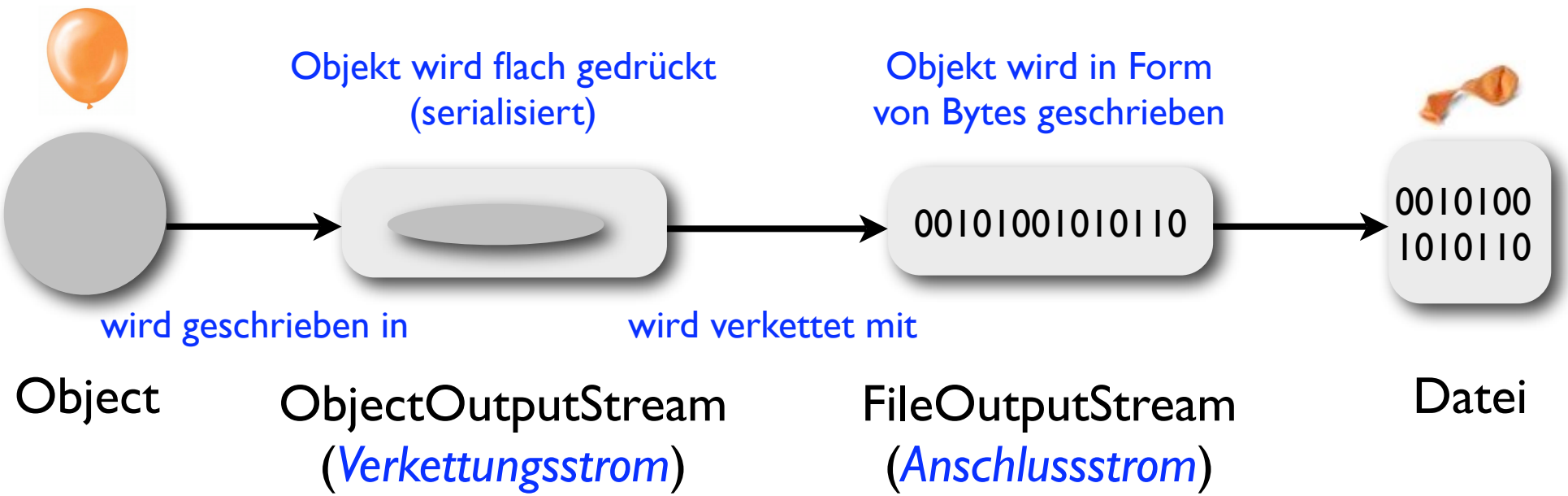
Was muß von einem Objekt gespeichert werden, damit es später wiederhergestellt werden kann?

Elementare Werte der Instanzvariablen.

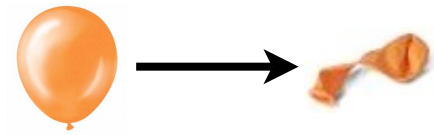
Objekte, die es über Instanzvariablen referenziert.

Informationen über die Klasse des Objekts (und die Klasse der referenzierten Objekte, aber nicht die Klasse selbst).

Objekte serialisieren



Objekte serialisieren



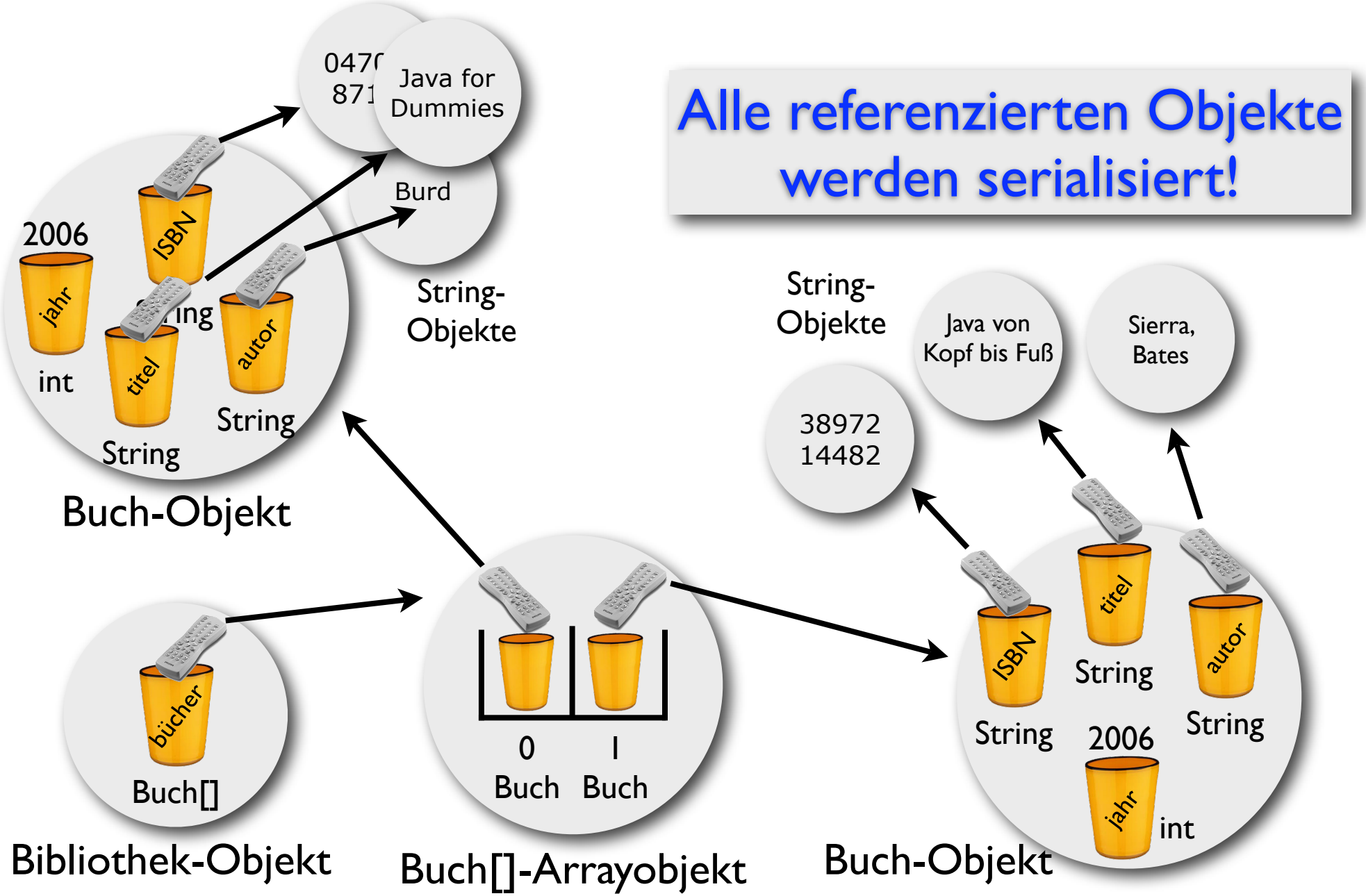
```
// Anschlussstrom verbindet sich zur Datei
FileOutputStream fileStream = new FileOutputStream ("Ballon.ser");

//  Verkettungsstrom verbindet sich mit dem Anschlussstrom
ObjectOutputStream os = new ObjectOutputStream (fileStream);

// Ballon als "serialisierbar" markieren
class Ballon implements Serializable { ... }

// zwei neue und bunte Ballon-Objekte speichern (serialisieren)
os.writeObject(new Balloon("Rot"));
os.writeObject(new Balloon("Orange"));

// Verkettungsstrom schliessen (inkl. Anschlussstrom)
os.close();
```

Beispiel: Serialisierung

```
import java.io.*;  // Paket für Datei-E/A

class Buch implements Serializable { // Buch wird "serialisierbar"
   String titel, autor, ISBN;      // Serialisierbar bedeutet, dass die Werte der
   int jahr;                       // Instanzvariablen gespeichert werden können.

   Buch (String t, String a, int j, String nr) {      // Konstruktor
      titel = t; autor = a; jahr = j; ISBN = nr;
   }
}

class Bibliothek implements Serializable {   // Bibliothek wird "serialisierbar"
   Buch[] bücher = new Buch[2];               // ein Array von Büchern
   ...
   void add (Buch einBuch) { ... }            // ein Buch hinzufügen
}
```

```
import java.io.*;  // Paket für Datei-E/A

class SerializableTest {
   public static void main(String[] args) {
      Bibliothek javaBib = new Bibliothek();  // ein neues Bibliothek-Objekt
      javaBib.add(new Buch("Java von Kopf bis Fuß","Sierra, Bates", 2006, ...));
      javaBib.add(new Buch("Java for Dummies","Burd", 2006, ...));
      ...
      try {  // E/A-Operationen können Exceptions auslösen
         FileOutputStream fs = new FileOutputStream ("buecher.ser");
         ObjectOutputStream os = new ObjectOutputStream (fs);
         os.writeObject(javaBib);  // Bibliothek-Objekt mit allen Büchern serialisieren
         os.close();
      } catch (IOException ex) {
         // auf Exceptions reagieren
      }
   }
}
```

Serialisierung im Detail



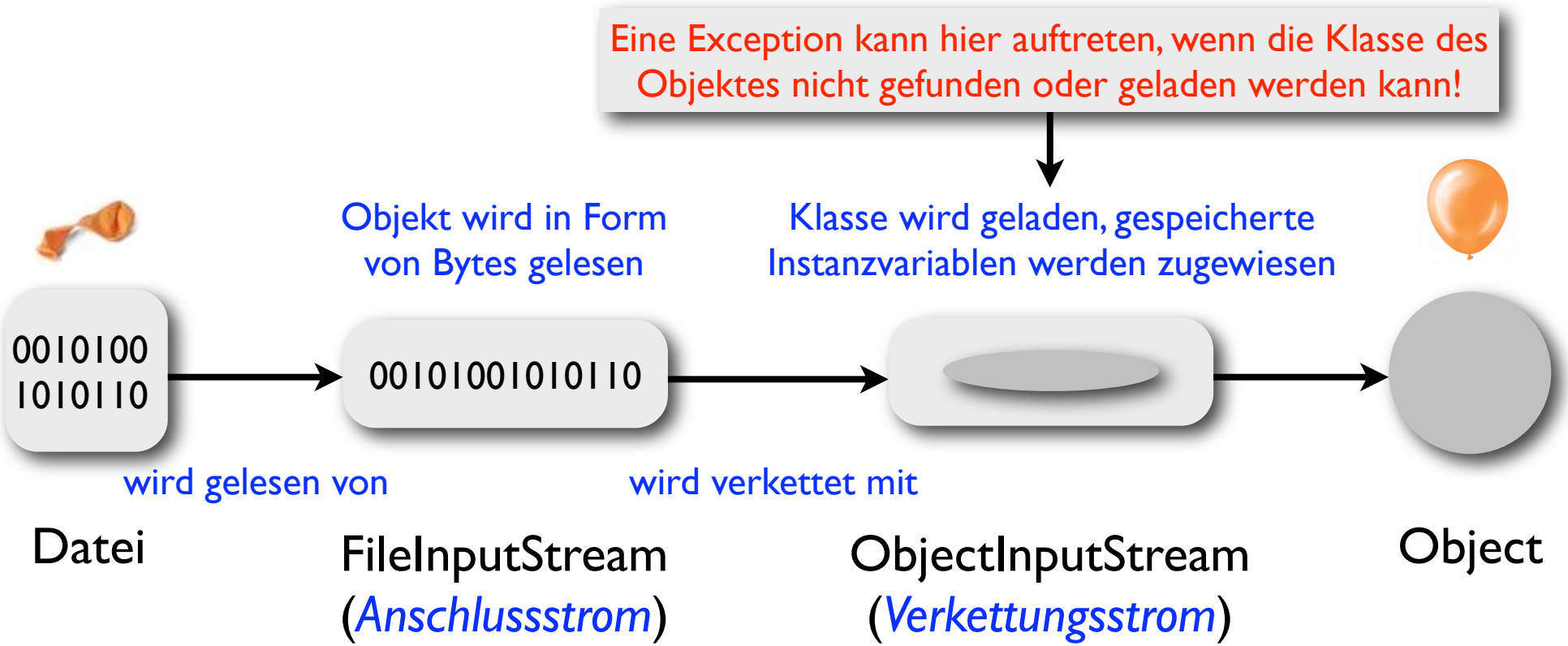
Geht das?


```
import java.io.*;

class Buch {                                          // Buch ist nicht serialisierbar!
    String titel, autor, ISBN;
    int jahr;
    ...
}

class Bibliothek implements Serializable {
    Buch[] bücher = new Buch[2];                      // Buch[] ist nicht serialisierbar!
    ...
}

class SerializableTest {
    ...
    Bibliothek javaBib = new Bibliothek();  // ein neues Bibliothek-Objekt
    javaBib.add(new Buch("Java von Kopf bis Fuß","Sierra, Bates", 2006, ...));
    ...
    os.writeObject(javaBib);  // das Bibliothek-Objekt serialisieren      // Exception !!!
    ...
}
```

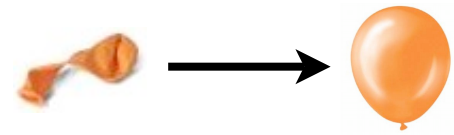
transient: Instanzvariable nicht serialisieren

```
class UserLogin implements Serializable {
   private String login;
   private transient String password;  // password nicht serialisieren !!!
}

// irgendwo im Programm
UserLogin user1 = new UserLogin();
...
FileOutputStream fs = new FileOutputStream ("accounts.ser");
ObjectOutputStream os = new ObjectOutputStream (fs);
os.writeObject(user1);                    // ok, keine Exception
os.close();
```

Objekte deserialisieren



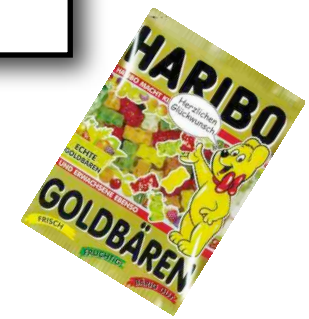
Objekte deserialisieren



```
// Anschlussstrom verbindet sich zur Datei
FileInputStream fileStream = new FileInputStream ("Ballon.ser");

//  Verkettungsstrom verbindet sich mit dem Anschlussstrom
ObjectInputStream os = new ObjectInputStream (fileStream);

// Objekte lesen (deserialisieren)
Object obj1 = os.readObject();  // Objekte in der gleichen Reihenfolge
Object obj2 = os.readObject();  // einlesen, in der sie gespeichert wurden.

// Objekte casten
Ballon ballon1 = (Ballon) obj1;
Ballon ballon2 = (Ballon) obj2;

//Verkettungsstrom schliessen (inkl. Anschlussstrom)
os.close();
```

```
class KlasseA {
    String artikel;
    KlasseA () { }
    KlasseA (String s) { artikel = s; }
}

class KlasseB extends KlasseA implements Serializable {
    int anzahl;
    transient String ID;
    KlasseB (String bezeichnung, int anz, String eineID) {
        super (bezeichnung);
        anzahl = anz; ID = eineID;
    }
}

// irgendwo im Programm
KlasseB einB = new KlasseB("Topf", 17,"xyz");
...
os.writeObject(einB);  // serialisieren
...
KlasseB neuesB = (KlasseB) is.readObject(); // deserialisieren
```

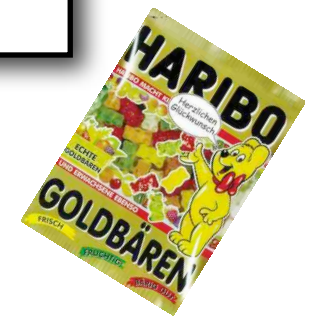
Welche Werte haben die Instanzvariablen *artikel, anzahl und ID* von **neuesB**?



SMS mit Ergebnis an:

```
class KlasseA {
    String artikel;
    KlasseA () { }
    KlasseA (String s) { artikel = s; }
}

class KlasseB extends KlasseA implements Serializable {
    int anzahl;
    transient String ID;
    KlasseB (String bezeichnung, int anz, String eineID) {
        super (bezeichnung);
        anzahl = anz; ID = eineID;
    }
}

// irgendwo im Programm
KlasseB einB = new KlasseB("Topf", 17,"xyz");
...
os.writeObject(einB);  // serialisieren
...
KlasseB neuesB = (KlasseB) is.readObject(); // deserialisieren
```

Welche Werte haben die Instanzvariablen *artikel, anzahl und ID* von **neuesB**?



artikel = null (Default-Konstruktor)
anzahl = 17 (serialisiert)
ID = null (transient)

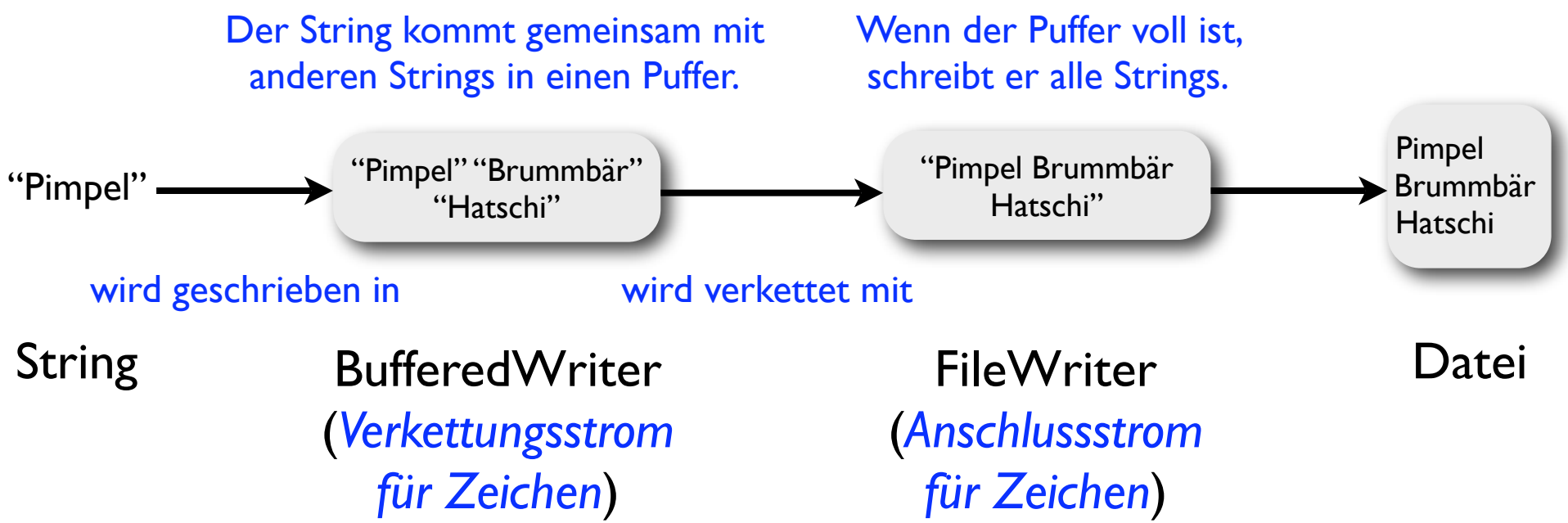
java.io.File (repräsentiert eine Datei, nicht ihren Inhalt)

```
File datei = new File ("text.txt");          // File-Objekt repräsentiert eine Datei
datei.createNewFile();                       // die Datei erzeugen
datei.getAbsolutePath();                     // den absoluten Pfadnamen erhalten

boolean existiert = datei.exist();           // existiert die Datei?
boolean lesbar = datei.canRead();            // ist die Datei lesbar?
boolean schreibbar = datei.canWrite();       // ist die Datei schreibbar?
long laenge = datei.length();                // Länge der Datei
boolean istGelöscht = datei.delete();        // Datei löschen

File dir = new File ("Java5");               //Verzeichnisnamen
dir.mkdir();                                 //Verzeichnis erstellen
if (dir.isDirectory()) {
   String[] inhalt = dir.list();             // Inhaltsverzeichnis auslesen
}
```

Textdateien schreiben

```
import java.io.*;

class SchreibeTextDatei {
   public static void main (String[] args) {
   try {
      // Textdatei öffnen (ggf. erstellen) und Strings schreiben
      FileWriter writer = new FileWriter(new File("buecher.txt"));
      writer.write ("Java von Kopf bis Fuß/Sierra, Bates/2006");
      writer.write ("Java for Dummies/Burd/2006");
      writer.close();  // Datei schliessen
   } catch (IOException ex) { ... }
}
```

Textdateien und Puffer



```
// Puffer als Verkettungsstrom zum Schreiben verwenden
File datei = new File ("buecher.txt");                    // Datei-Objekt
FileWriter fileWriter = new FileWriter (datei);
BufferedWriter writer = new BufferedWriter (fileWriter);  // Puffer
writer.write ("Java von Kopf bis Fuß/Sierra, Bates/2006");
writer.newLine();                                         // \n ausgeben
writer.write ("Java for Dummies/Burd/2006");
writer.newLine();
//writer.flush();                                         // alle Daten aus dem Puffer jetzt schreiben
writer.close();                                           // der Puffer schliesst auch den FileWriter
```

```
// Puffer als Verkettungsstrom zum Lesen verwenden
FileReader fileReader = new FileReader (new File("buecher.txt"));
BufferedReader reader = new BufferedReader (fileReader);  // Puffer
String zeile = "";  // hier wird die gelesene Eingabezeile gespeichert
while ((zeile = reader.readLine()) != null) {  // einlesen, solange Daten existieren
   System.out.println(zeile);
}
reader.close();  // der Puffer schliesst auch den FileWriter
```

Strings parsen mit split()

```
// Bücherinformationen in die Datei schreiben
// einen Schrägstrich "/" zur Trennung der Werte benuzten
writer.write ("Java von Kopf bis Fuß/Sierra, Bates/2006");
...
FileReader fileReader = new FileReader (new File("buecher.txt"));
BufferedReader reader = new BufferedReader (fileReader);
String zeile = "";  // hier wird die gelesene Eingabezeile gespeichert

// alle Zeilen aus der Datei einlesen und aufbrechen
while ((zeile = reader.readLine()) != null) {
   String[] ergebnis = zeile.split("/");  // den String zerlegen, Trennzeichen "/"
   for (String token : ergebnis) System.out.println(token);
}
reader.close();
```

Jetzt sind Sie wieder an der Reihe!

Lesen Sie zu Serialisierung und Datei-E/A **Kapitel 14**.