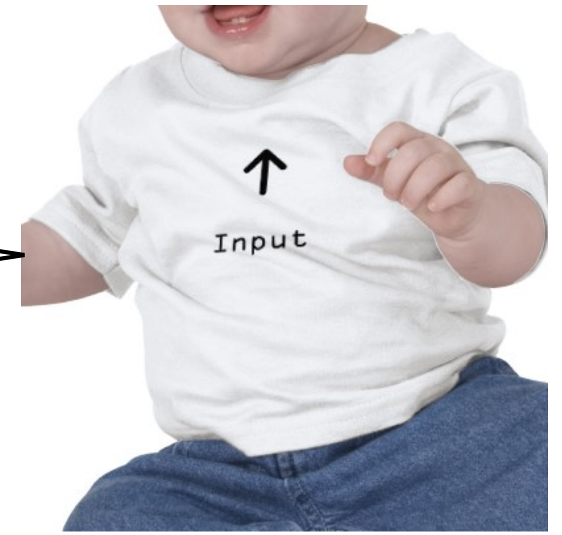
Quiz: Wo ist “Output”?



http://www.zazzle.de/input_output_tshirt-235248123204113601

Ein- und Ausgabe

OOPM, Ralf Lämmel

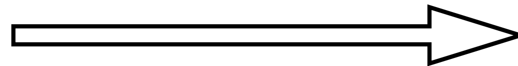
Java Plattform = Sprache + APIs (Bibliotheken) + ...

* java.applet	Applet-Programmierung
* java.awt	GUI-Programmierung mit AWT
* java.beans	JavaBeans™-Entwicklung
* java.io	Ein-/Ausgabe, Datenströme
* java.lang	fundamentale Klassen (z.B. String)
* java.net	Netzwerkfunktionen
* java.rmi	Remote Method Invocation
* java.security	Zertifikate, Kryptographie
* java.sql	Datenbank-Funktionen
* java.util	Klassen für Datenstrukturen
* javax.swing	GUI-Programmierung mit Swing
* org.w3c.dom	XML-Programmierung mit DOM
* ...	

Das Konzept der **Ströme**

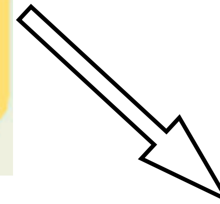
Programm

Schreiben

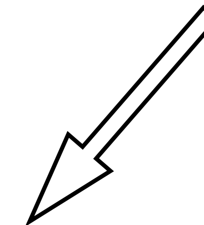


in Datensenke

Datei



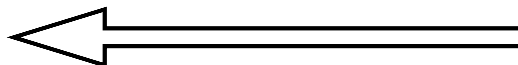
Programm



Datei

Programm

Lesen



aus Datenquelle

Ströme (Streams)

* ***Sequentiell*** organisierte Datenmengen
  * Auf untersten Ebene
    * ***Byte***folgen
    * ***Zeichen***folgen
  * Auf höherer Ebene
    * Primitive Werte
    * Serialisierte Objekte
  * mit Zugriffsformen
    * lesend (“Datenquellen”)
    * schreibend (“Datensenken”)
* Ausnahme: RandomAccessFile
  * Wahlfreier Zugriff auf Datei von Bytes

Beispiel: Addiere 2 Werte aus einer Datei!

```
Reader r = ...;

int value1 = Integer.parseInt(r.readLine());
int value2 = Integer.parseInt(r.readLine());
int sum = value1 + value2;
System.out.println(value1 + " + " + value2 + " = " + sum);
r.close();
```

Illustrationen zur Ein-/Ausgabe
siehe Repository: oo.file

Aufgaben von *strom*basierter I/O

* Datenaustausch / Kommunikation
  * zwischen Programm und Nutzer
  * zwischen Programm und Netzwerk
  * zwischen Programmen
* Dateizugriff im Dateisystem
* Persistenz von Objekten
* Remote Method Invocation

Bibliotheksunterstützung für Ströme

(*Basis*-) Klassen für Ströme

* File	Datei- und Verzeichnisnamen
* *InputStream*	**Byte**strom zum Lesen
* *OutputStream*	**Byte**strom zum Schreiben
* RandomAccessFile	Wahlfreier Zugriff auf Dateien mit **Bytes**
* *Reader*	**Zeichen**strom zum Lesen
* *Writer*	**Zeichen**strom zum Schreiben
* ...	

Wichtige Arten von ***Eingabe***strömen

* File
* InputStream
  * ByteArrayInputStream — Lies von einem Feld von Bytes
  * FileInputStream — Lies Bytes von einer Datei
  * FilterInputStream — Filtere (transformiere) einen Strom
  * ObjectInputStream — Serialisierung von Objekten
  * PipedInputStream — Erlaubt Modus vergleichbar zu “|” in Unix
  * SequenceInputStream — Verkettung zweier Ströme
  * StringBufferInputStream — Lies Bytes von einem String
* OutputStream
* RandomAccessFile
* Reader
* Writer

Konstruieren von Strömen ausgehend von einer Datei (File) f

Lies Bytes

Lies Zeichen

... mit Puffer

- fis = **new** *FileInputStream*(f)
  - isr = **new** *InputStreamReader*(fis)
    - br = **new** *BufferedReader*(isr)
- fos = **new** *FileOutputStream*(f)
  - osw = **new** *OutputStreamWriter*(fos)
    - bw = **new** *BufferedWriter*(osw)

Beispiel: Addiere 2 Werte aus einer Datei!

```
public class Program {

   public static void main(String[] args) throws IOException {
      File input = new File(args[0]);
      BufferedReader br = new BufferedReader(
            new InputStreamReader(
               new FileInputStream(input)));
      int value1 = Integer.parseInt(br.readLine());
      int value2 = Integer.parseInt(br.readLine());
      int sum = value1 + value2;
      System.out.println(value1 + " + " + value2 + " = " + sum);
      br.close();
   }
}
```

Die Eingabe wird mittels "Dateieingabestrom" gelesen.

Illustrationen zur Ein-/Ausgabe siehe Repository: oo.file

Schreiben der ***Ausgabe*** in eine Datei

```
// Write the output file with a buffered writer
BufferedWriter bw = new BufferedWriter(
          new OutputStreamWriter(
            new FileOutputStream(output)));

// Write result
bw.write(value1 + " + " + value2 + " = " + sum);

// Done with the output
bw.close();
```

Ausnahmebehandlung

```
public class Program {

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

      // Lese 2 Zahlen aus Datei und addiere sie.

   }
}
```

Ausnahmebehandlung

```
try {

  ... code as before ...

} catch (FileNotFoundException fnfe) {
   System.err.println(
      "Input file missing");
} catch (IOException ioe) {
   System.err.println(
      "Some I/O error");
} catch (NumberFormatException nfe) {
   System.err.println(
      "Content of input file cannot be converted");
}
```

Die Schnittstelle von InputStream

- Wichtigste Methoden
  - *read*(): Lies ein **Byte**.
  - *close*(): Strom schließen.
- Weitere Methoden
  - *available*(...): Teste ob nächstes read(...) blockiert.
  - *read*(...): Lies **Bytes** in ein Feld.
  - *skip*(...): Überlese eine gewisse Anzahl **Bytes**.
  - Optional
    - *mark*(...): Merke die aktuelle Position vor.
    - reset(...): Kehre zu früherer Position zurück.

Die Schnittstelle von Reader

- Wichtigste Methoden
  - *read*(): Lies ein **Zeichen.**
  - *close*(): Strom schließen.
- Weitere Methoden
  - *ready*(): Teste ob nächstes read(...) blockiert.
  - *read*(...): Lies **Zeichen** in ein Feld.
  - *skip*(...): Überlese eine gewisse Anzahl **Zeichen**.
  - Optional
    - *mark*(...): Merke die aktuelle Position vor.
    - reset(...): Kehre zu früherer Position zurück.

Das ist InputStream modulo Umbennung.

BufferedReader

- Benutzt Zwischenspeicher (Puffer)
- Optimierungen
  - Anzahl der Zugriffe auf den Eingabestrom.
  - Anzahl der Konvertierungen und Ergebnisse.
- Wichtige zusätzliche Methode
  - *readLine*(): Liest eine ganze Zeil als String ein.

Beispiel: Unix’ more

* Eingabe
  * Kommandozeilenarrgument
    * Name einer Textdatei
* Ausgabe
  * Standardausgabe
    * Inhalt der Datei

Illustrationen zur Ein-/Ausgabe
siehe Repository: oo.file

```java
public class More {
  public static void main(String[] args)
              throws IOException {
    // Obtain file name from command line
    File file = new File(args[0]);
    // Construct a buffered reader for the input
    BufferedReader br = new BufferedReader(
          new InputStreamReader(
              new FileInputStream(file)));
    // Read from the input line by line and echo
    String line;
    while ((line = br.readLine()) != null) {
      System.out.println(line);
    }
    // Done
    br.close();
  }
}
```

Die Schnittstelle von Writer

* Wichtigste Methoden
  - *write*(...): Schreibe ein **Zeichen**.
  - *close*(): Strom schließen.
  - *flush*(): Eventuell gepufferte **Zeichen** schreiben.

URLs und I/O

- Klasse java.net.URL
  - Konstruktor
    - *URL*(String s)
  - Zuordnung eines Eingabestroms
    - *openStream*()

Lesen aus einem URL

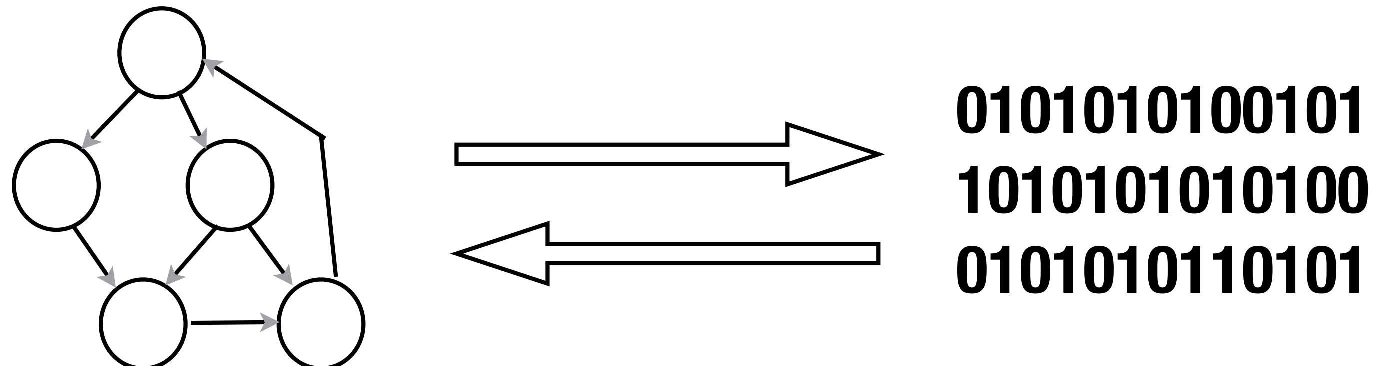
```
// Obtain URL from command line
URL url = new URL(args[0]);

// Construct a buffered reader for the input
BufferedReader br =
   new BufferedReader(        // Make the reader a buffered reader
     new InputStreamReader(   // Create a reader from the stream
        url.openStream()));   // Create an input stream from URL
```

Ein- und Ausgabe auf höherer Ebene für den Zweck der ***Serialisierung von Objekten***

Persistente Objekte (bzw. Serialisierung)

- Definition
  - *Objekte bleiben über Programmlaufzeit erhalten*
- Eine Vorgehensweise
  - Serialisierung der Objekte vor Programmende
  - Speicherung des Zustandes in Datei oder Datenbank
  - Wiederherstellung der Objekte in anderem Programmlauf
- Siehe Schnittstelle java.io.Serializable



A serialized object

0101010010110100101010010101010101010101001010
01101010101001010100101010100101001001101001

A class of serializable objects

```
public class Employee implements Serializable {
  private static final long serialVersionUID = -210889592677165250L;

  private Person person;
  private double salary;

  public Person getPerson() { return person; }
  public void setPerson(Person person) { this.person = person; }
  public double getSalary() { return salary; }
  public void setSalary(double salary) { this.salary = salary; }
}
```

Eindeutige Identifikation der Klasse insbesondere auch für die Serialisierung

Serialize an object

Ein Objekt eines serialisierbaren Betriebes mit serialisierbaren Angestellten etc.

```
Company sampleCompany = new Company();
...
FileOutputStream fos = new FileOutputStream(filename);
ObjectOutputStream out = new ObjectOutputStream(fos);
out.writeObject(sampleCompany);
out.close();
```

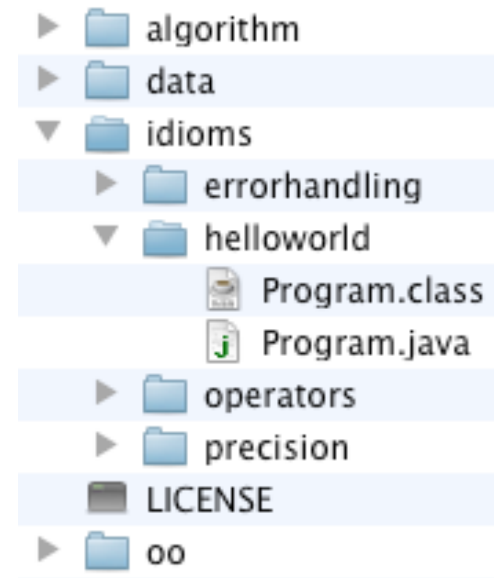
De-serialize an object

```
Company sampleCompany;
FileInputStream fis = new FileInputStream(filename);
ObjectInputStream in = new ObjectInputStream(fis);
sampleCompany = (Company)(in.readObject());
in.close();
```

Zugriff auf das Dateisystem

- Testen von Datei-Eigenschaften
- Löschen von Dateien
- Anlegen von Verzeichnissen
- Lesen von Verzeichnissen

Hierarchisches Dateisystem (modulo Links)



Testen von Datei-Eigenschaften (Klasse java.io.File)

* boolean *exists*()	Existiert die Datei mit dem gegebenen Namen?
* boolean *canRead*()	Ist die Datei lesbar? Rechte verfügbar?
* boolean *canWrite*()	Ist die Datei schreibbar? Rechte verfügbar?
* boolean *isFile*()	Ist diese Datei tatsächliche eine Datei?
* boolean *isDirectory*()	... oder ist es ein Verzeichnis?
* boolean *isHidden*()	Ist diese Datei “verborgen”?
* long *length*()	Was ist die Länge der Datei?

Änderungen im Dateisystem (Klasse java.io.File)

* boolean *delete*() Lösche die Datei.
* boolean *renameTo*(...) Benenne die Datei um.
* void *deleteOnExit*() Lösche die Datei bei Programmende.
* boolean *mkDir*() Lege ein Verzeichnis an.
* boolean *mkDirs*() ... einschließlich aller Unterverzeichnisse.

Verzeichnisse lesen (Klasse java.io.File)

- File[] *listFiles*() — Liste Dateien im Verzeichnis.
- File[] *listFiles*(...) — Filtere die Liste zusätzlich.

Beispiel: Wir wollen nun rekursiv alle Dateien ausgehend von einem Wurzelverzeichnis aus auflisten.

```
public static void listDirectories(File dir) {
    File[] children = dir.listFiles(
                         new NotHiddenFilter());
    if (children != null) {
        for (int i=0; i<children.length; i++) {
            File file = children[i];
            System.out.println(file.toString());
            if (file.isDirectory())
               listDirectories(file);
        }
    }
}
```

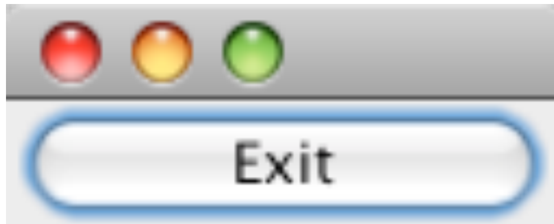
```java
/**
 * A filter that accepts all non-hidden files.
 */
public class NotHiddenFilter implements FileFilter {
    public boolean accept(File file) {
        return !file.isHidden();
    }
}
```

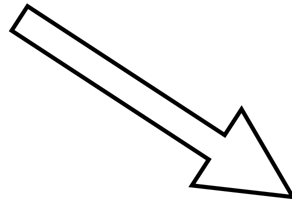
Graphische Benutzeroberflächen

- *Eingabe*: Nutzereingabe
- *Ausgabe*: GUI-Sichten
- Wichtige Aspekte:
  - Verwaltung der Daten in einem *Modell* (Objekten)
  - Registrierung von Routinen zur *Ereignisbehandlung*
  - Konstruktion der GUI-Komponenten

(C) 2007-2018, Ralf Lämmel, Universität Koblenz-Landau

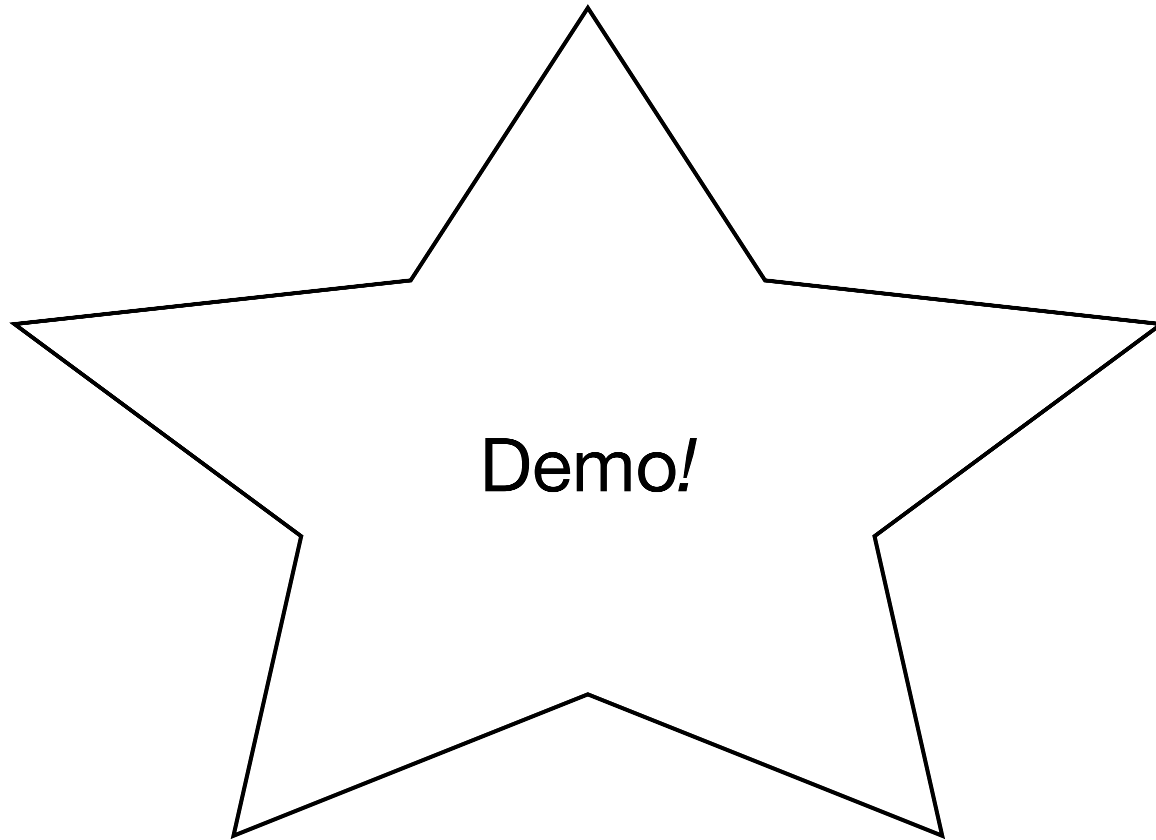
Beispiel: “Exit” Funktion einer Applikation





```
System.out.println("Exiting ...");
System.exit(0);
```

Siehe Repository: oo.gui


Demo!

Bestandteile der “Exit”-Applikation

1. Konstruktion der GUI-Komponenten
2. Definition der Ereignisbehandlungsroutine
3. Handlerregistrierung bei der GUI
4. Aktivierung (Anzeigen) der GUI

Konstruktion der GUI-Komponenten

```
import javax.swing.*;

...
JFrame f = new JFrame();
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JButton b = new JButton("Exit");
f.add(b);
f.pack();
...
```

Java's "GUI API"

Aus den Java-Tutorials: "A Frame is a top-level window with a title and a border."

Aus den Java-Tutorials: "A common (push) button."

Aus der Java-Dok.: "Appends the specified ***component*** to the end of this ***container***."

Aus der Java-Dok.: "Causes this ***window*** to be sized to fit the preferred size and layouts of its subcomponents."

Definition der Ereignisbehandlungsroutine

```
import java.awt.event.*;
```

Java's API für Ereignisbehandlung und "Low-Level" GUI-Funktionalität

```
public class ExitAction implements ActionListener {
   public void actionPerformed(ActionEvent event) {
      System.out.println("Exiting ...");
      System.exit(0);
   }
}
```

Aus der Java-Dok.: "The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked."

Handlerregistrierung bei der GUI

das Button-Objekt

Schnittstellen-polymorphe Methode zur Registrierung

```
b.addActionListener(
  new ExitAction());
```

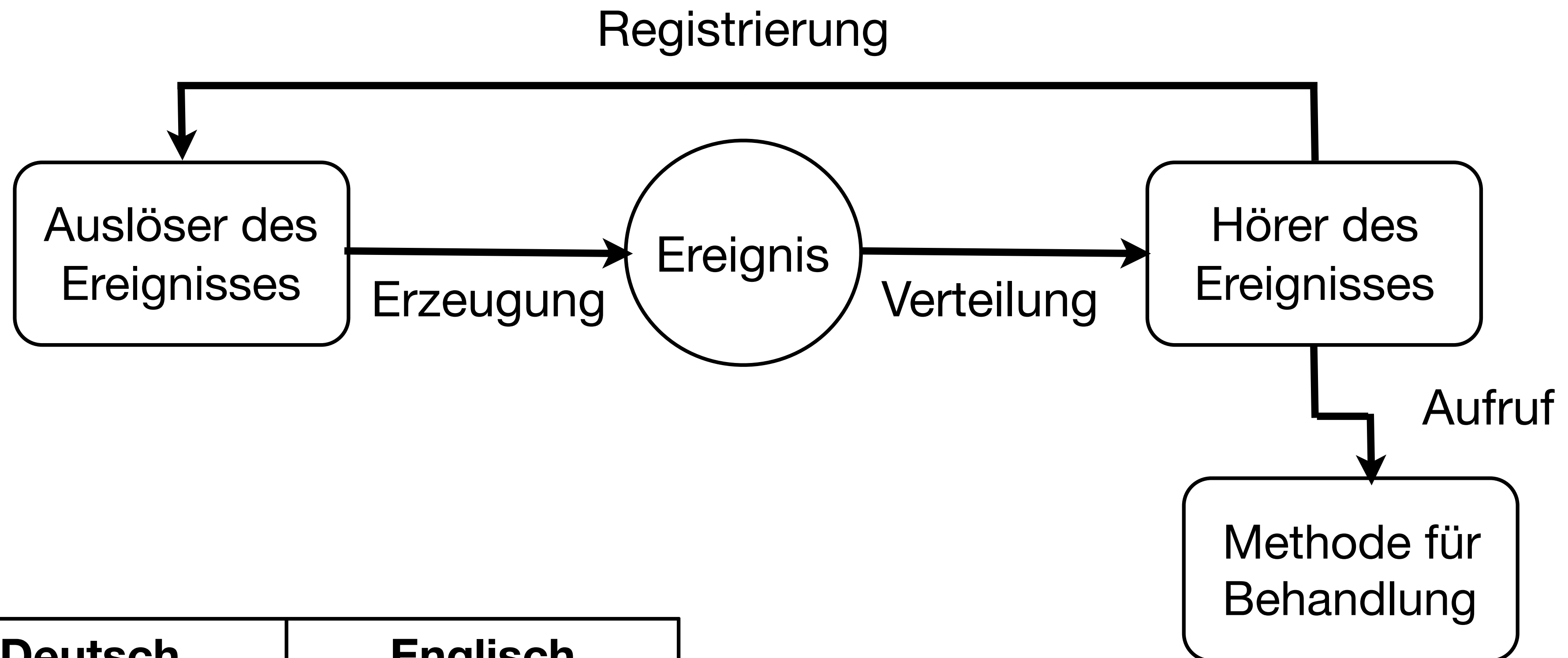
Objekt welches die Behandlung bietet

Aktivierung (Anzeigen) der GUI

```
f.setVisible(true);
```

Die GUI ist fertig konstruiert und konfiguriert. Damit kann sie projiziert werden und auf Ereignisse reagieren.

Grundidee der Ereignisbehandlung



Deutsch	Englisch
Ereignis	Event
Auslösen	To trigger
Zuhörer	Listener
Behandler	Handler

Zusammenfassung des allgemeinen Ereignis-Zyklus

* ***Erzeugung***: eine Quelle (ein Objekt) erzeugt ein Objekt als Manifestation des Ereignisses.
* ***Verschicken***: die Quelle (oder eine der Quelle bekannten Autorität) versendet das Ereignis an alle (potentiell) interessierten Zuhörer.
* ***Verarbeitung***: der Zuhörer empfängt das Ereignis und entscheidet über die Verarbeitung.
* ***Abschluss***: das Ereignis ist konsumiert wenn alle Zuhörer ihre Verarbeitung beendet haben.

Begrifflichkeiten bei der “Exit”-Applikation

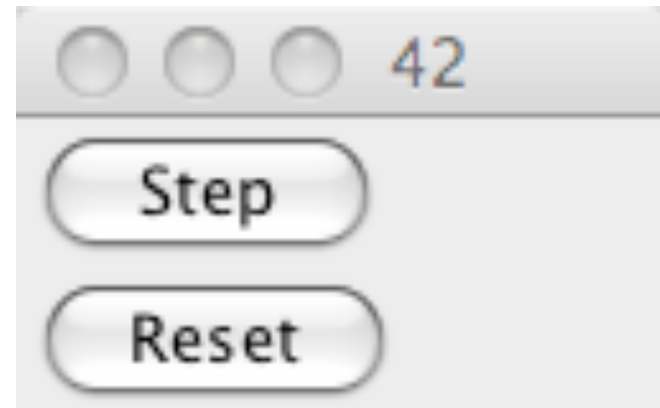
* *Auslöser des Ereignisses*
  - GUI der Applikation (Nutzer wählt “Exit” Button)
* *Ereignis*
  - Aktion “button pressed”
* *Zuhörer*
  - Objekt mit entsprechender Schnittstelle
* *Behandlung*
  - System.exit(0)

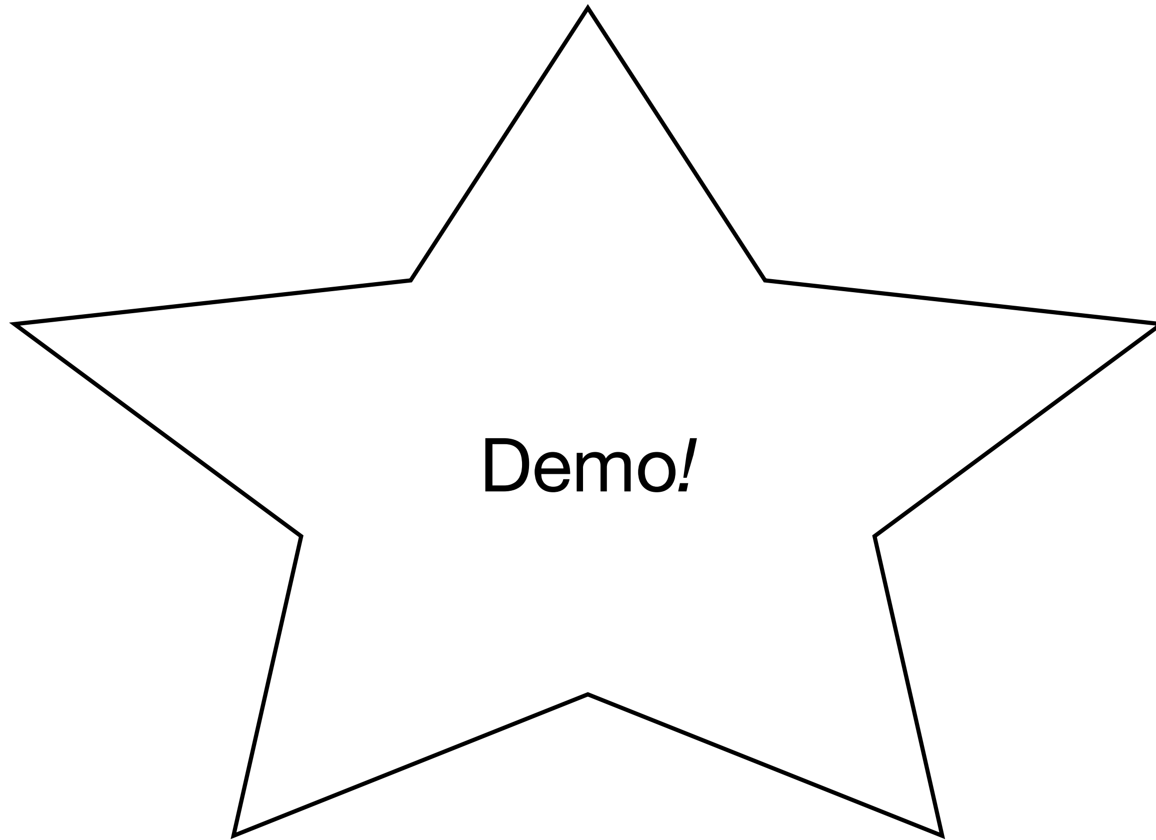
Verallgemeinerte Ereignisse in einer OO GUI

* ***Ereignis***: Auftreten einer Eingabe-basierten Situation in der GUI
* ***Ereignis-Quelle***: GUI-Komponente die mit dem Ereignis assoziiert ist
* ***Ereignis-Objekt***: OO-Repräsentation des Ereignisses
* ***Ereignis-Zuhörer***: Objekte die über Ereignisse benachrichtigt werden wollen.
* ***Ereignis-Aktion***: Methode (Handler) welche der Zuhörer zur Ausführung bringt

Komplexeres Beispiel: GUI-Interaktion mit einem Modell

- Modell
  - Daten der Applikation
    - Ein Zähler-Objekt
- GUI
  - “Sicht auf Modell” (View)
    - Anzeige
      - Zählerstand
    - Aktionen
      - Step: Zähler erhöhen
      - Reset: Zähler zurücksetzen




Demo!

Konstruktion der GUI-Komponenten mit Registrierung und Start der GUI

```
frame = new JFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
updateTitle();
panel = new JPanel();
frame.add(panel);
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
step = new JButton("Step");
step.addActionListener(this);
panel.add(step);
reset = new JButton("Reset");
reset.addActionListener(this);
panel.add(reset);
frame.pack();
frame.setVisible(true);
```

Container zur Anreihung von Komponenten

Vertikale Anreihung der Komponenten

Spezialfall:
Konstruierendes Objekt
= Behandelndes Objekt

Eine Ereignisbehandlungsroutine für beide Buttons

```
public void actionPerformed(ActionEvent event) {
   if (event.getSource() == step) {
      counter.step();
      updateTitle();
      return;
   }
   if (event.getSource() == reset) {
      counter.reset();
      updateTitle();
      return;
   }
}
```

Trennung von Modell und Sicht

```
public class View implements ActionListener {
   private Counter counter;
   private JFrame frame;
   private JPanel panel;
   private JButton step;
   private JButton reset;
   public View(Counter counter) {
   ... store counter and construct GUI ...
   }
   public void actionPerformed(ActionEvent event) {
      ... handle the GUI event ...
   }
   public void updateTitle() {
      frame.setTitle(Integer.toString(counter.getCount()));
   }
}
```

Eine einzelne Klasse übernimmt die Aufbewahrung der GUI-Komponenten und des Anwendungszustandes ("Counter") sowie die Ereignisbehandlung ("implements ActionListener").

GUI-Komponenten (Swing API)
Die javax.swing.JComponent-Hierarchie

* Im Beispiel
  - JFrame — Eine umrahmtes Fenster
  - JPanel — Container für Komponenten mit Layout
  - JButton — Ein Button
* Weitere
  - JLabel — Textkomponente
  - JList — Auswahl aus einer Liste
  - JTextField — Eingabefeld
  - JComboBox — Kombination aus Feld und Liste
  - JSlider — Wertauswahl mit Slider
  - ...

Ereignisse in interaktiven Programmen

- ***High-level*** Ereignisse
  - Fenster schließen/öffnen
  - Button gedrückt
  - Editierbarer Text geändert
- ***Low-level*** Ereignisse
  - Tastatur-Ereignisse
  - Mouse-Ereignisse
  - Timer-Ereignisse
  - Peripherie-Ereignisse

GUI-Ereignisse: Klassenhierachie in java.awt.event

* *AWTEvent*
  - ActionEvent — High-level, Komponenten-spezifische Ereignisse
  - AdjustmentEvent — Ereignisse für anpassbare Komponenten
  - ComponentEvent — Low-level, Komponenten-spezifische Ereignisse
    - FocusEvent
    - InputEvent
      - MouseEvent
        - MouseWheelEvent
      - KeyEvent
    - PaintEvent
    - WindowEvent
  - ItemEvent — Selektieren / Deselektieren eines Items
  - TextEvent — Änderung des Textes eines Objektes

MouseEvent

Method Summary	
int	**getButton**() Returns which, if any, of the mouse buttons has changed state.
int	**getClickCount**() Returns the number of mouse clicks associated with this event.
static String	**getMouseModifiersText**(int modifiers) Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".
Point	**getPoint**() Returns the x,y position of the event relative to the source component.
int	**getX**() Returns the horizontal x position of the event relative to the source component.
int	**getY**() Returns the vertical y position of the event relative to the source component.
boolean	**isPopupTrigger**() Returns whether or not this mouse event is the popup menu trigger event for the platform.
String	**paramString**() Returns a parameter string identifying this event.
void	**translatePoint**(int x, int y) Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.

Zuhörer (Schnittstellen) für GUI events

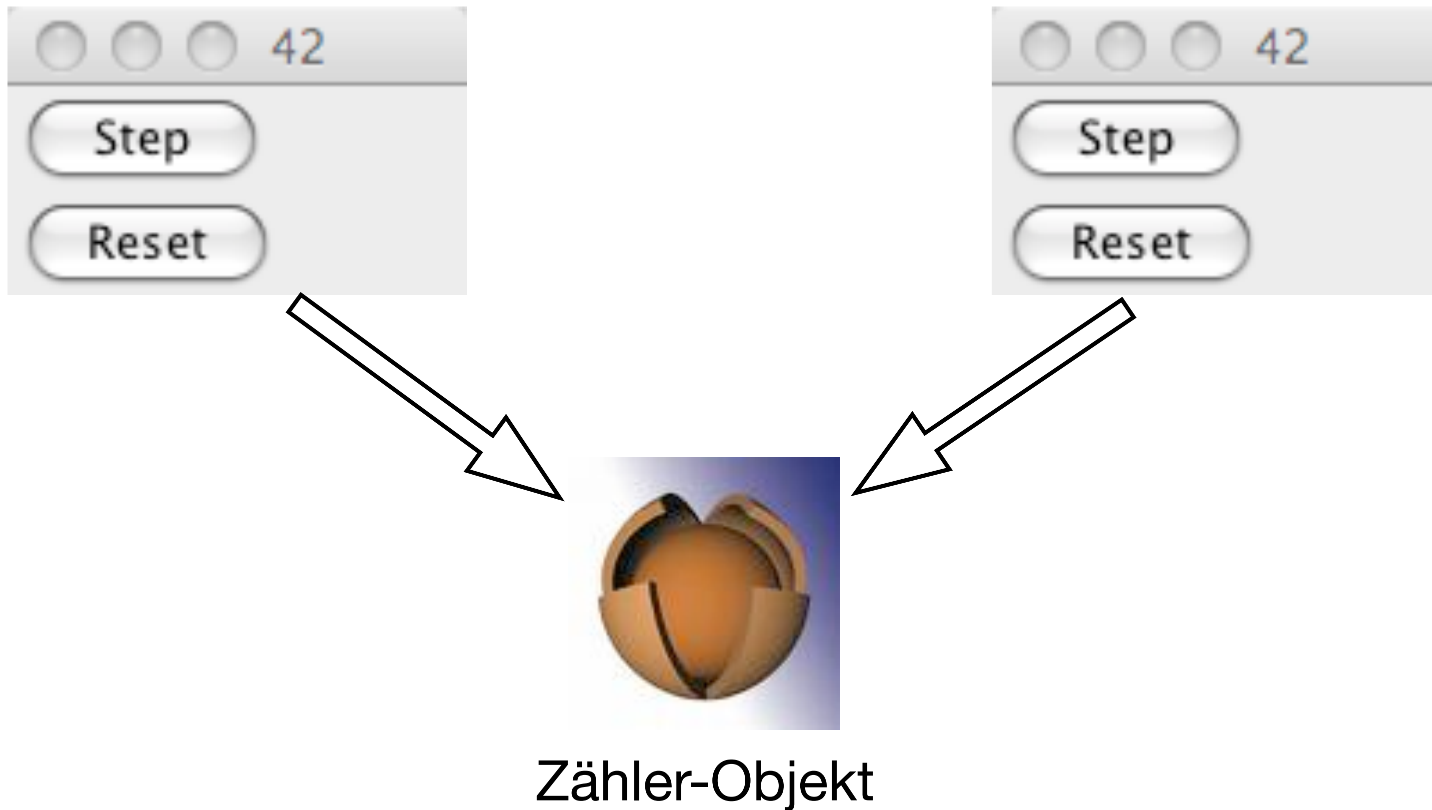
Ereignis	**Zuhörer**
ActionEvent	ActionListener
MouseEvent	MouseListener
...	...

(C) 2007-2018, Ralf Lämmel, Universität Koblenz-Landau

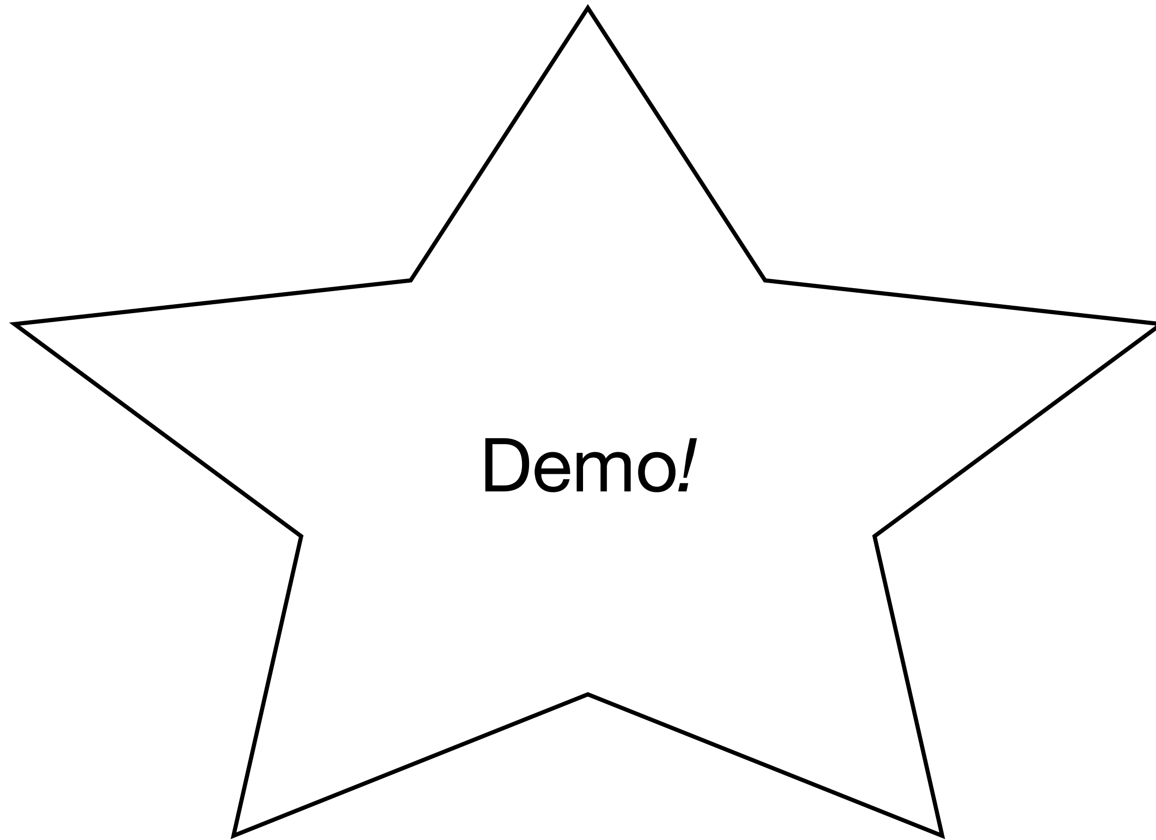
MouseListener

Method Summary	
void	mouseClicked(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	mouseEntered(MouseEvent e) Invoked when the mouse enters a component.
void	mouseExited(MouseEvent e) Invoked when the mouse exits a component.
void	mousePressed(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	mouseReleased(MouseEvent e) Invoked when a mouse button has been released on a component.

Mehrere GUI-Sichten auf ein Modell



Zähler-Objekt



(C) 2007-2018, Ralf Lämmel, Universität Koblenz-Landau

Ereignisse zur Zustandsänderung von Objekten

- Klasse *PropertyChangeEvent*
  - Konstruktorparameter
    - *Object source*
    - *String propertyName*
    - *Object oldValue*
    - *Object newValue*
- Schnittstelle *PropertyChangeListener*
  - *void propertyChange(PropertyChangeEvent e)*

Zähler-Objekte sollten nun alle abhängigen Sichten über Zustandsänderungen informieren.

Zähler-Klasse mit Protokoll für die Benachrichtigung bei Zustandsänderungen

```
public final class Counter {

    private int count = 0;

    private Collection<PropertyChangeListener> listeners =
        new LinkedList<PropertyChangeListener>();

    public void step() { setCount(count+1); }
    public void reset() { setCount(0); }
    public int getCount() { return count; }
    private void setCount(int value) { ... } // see next slide

    public void addListener(PropertyChangeListener listener) {
        listeners.add(listener);
    }
}
```

Verwaltung der Beobachter

Eintragen eines Beobachters

Setter mit Benachrichtigung

```
private void setCount(int newValue) {

    // Backup value
    int oldValue = count;

    // Do the update
    count = newValue;

    // Construct the event
    PropertyChangeEvent event =
        new PropertyChangeEvent(
            this,
            "count",
            oldValue,
            newValue);

    // Notify all subscribed listeners
    for (PropertyChangeListener l : listeners)
        l.propertyChange(event);
}
```

Verdrahten der Sichten und des Modells

```
public static void main(String[] args) {
    Counter counter = new Counter();
    new View(counter,100,100);
    new View(counter,200,150);
}
```

* Die Sichten müssen ...
  - sich mit dem Modell registrieren und
  - PropertyChangeListener implementieren.

Sichten als Beobachter

```
public class View
      implements ActionListener, PropertyChangeListener {

   private Counter counter;
   private JFrame frame;
   ...

   public View(Counter counter, int x, int y) {
      counter.addListener(this);
      ...
   }

   public void propertyChange(PropertyChangeEvent event) {
      if (event.getSource() == counter
      && event.getPropertyName().equals("count"))
     ...
   }
}
```

GAME OVER

- **Zusammenfassung**
  - Ein- und Ausgabe mit Strömen
  - Graphische Benutzeroberflächen
  - Ereignisse und Ereignisbehandlung
- **Ausblick**
  - Prüfungsvorbereitung
  - Prüfung
  - Mehr Kurse

(C) 2007-2018, Ralf Lämmel, Universität Koblenz-Landau