

# Zeichenketten

## 1. Bezug zum Unterricht: Anwendungen

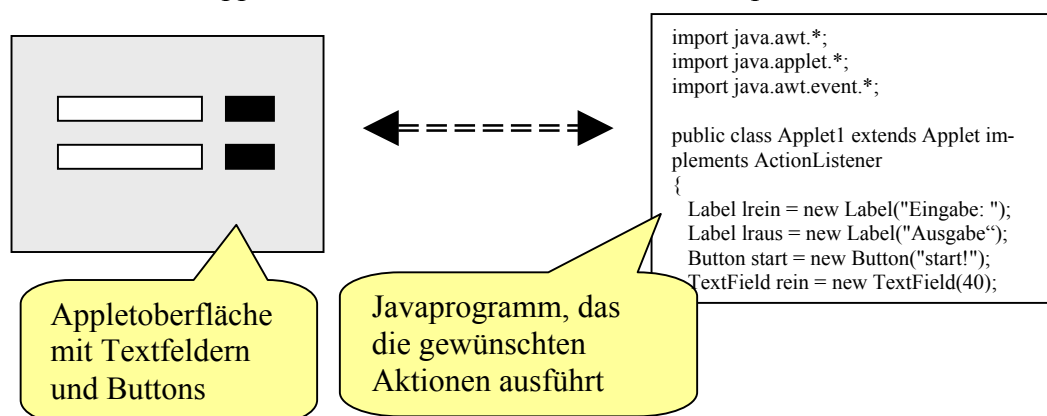
Die im Unterricht benutzten Elemente einer Programmiersprache sollten sich nicht aus Vollständigkeits- oder anderen fachimmanenten Überlegungen ergeben, sondern möglichst aus einer Problemstellung abgeleitet werden, die Bezug zu den Möglichkeiten und Folgen der **Anwendung von Informatiksystemen** hat. Da außer bei grafischen Problemen fast die gesamten Ein- und Ausgabe von Programmen über Zeichenketten erfolgt, ergeben sich entsprechend viele Anwendungsmöglichkeiten:

- Reine Datenverarbeitung (*Daten einlesen, Typkonvertierungen, verarbeiten, ausgeben*)
- Verschlüsselung (*Sicherheit, eCommerce, Vertrauen im Netz, ...*)
- Symbolverarbeitung (*Computeralgebra, DNA-Strukturen, mathematische Anwendungen...*)
- Textinterpretation (*Syntax, Mustererkennung, Übersetzung, Ersetzung, Suchvorgänge, ...*)

Nebenbei bietet die Zeichenkettenverarbeitung gute Möglichkeiten zur **Binnendifferenzierung** und eine Fülle von Aufgaben zur **Leistungsmessung**, die von sehr einfachen Fällen für den Anfangsunterricht bis zu ziemlich kniffligen Problemen reichen.

## 2. Oberfläche und Ereignisverarbeitung

Zeichenketten werden innerhalb des Java-Programms verarbeitet. Um Ausgangsmaterialien einzulesen und Ergebnisse darzustellen, benötigt man eine Verknüpfung mit Darstellungskomponenten des Bildschirms, die ziemlich beliebig ist. Wir wollen uns hier zur Darstellung auf Textfelder in einem Applet beschränken und Aktionen mit Knöpfen auslösen.



Die Arbeitsweise solcher Programme ist immer gleich:

- zuerst wird die grafische Oberfläche (GUI) erzeugt und angezeigt,
- dann wartet das Applet auf Eingaben, z. B. auf das Eintippen eines Textes in ein Textfeld, (schon dabei werden jede Menge Ereignisse (Events) ausgelöst, z. B. beim Loslassen einer Taste, die wir hier aber ignorieren wollen)

- beim Anklicken eines Knopfes (Buttons) wird ein Ereignis ausgelöst, das im Programm den gewünschten Ablauf startet. Dazu holt sich das Programm die erforderlichen Daten aus den GUI-Komponenten, verarbeitet sie und stellt die Ergebnisse auf diesen oder anderen GUI-Komponenten dar.

## 2.1 Die Erzeugung von GUI-Komponenten

Wir benötigen von den GUI-Komponenten des *AWT* (*abstract windowing toolkit*) von Java nur *Label*-, *TextField*- und *Button*-Objekte. Wir erzeugen sie mit Hilfe eines Konstruktors, dem oft die Aufschrift der Komponente oder deren Größe übergeben wird.

```
Label lrein = new Label("Eingabe:");
```

In der *Init*-Methode des Applets werden dann die Komponenten ggf. durch die *setBounds*-Methode mit Werten versehen (beim *null*-Layout) und dem Applet mit der *add*-Methode hinzugefügt. Ein Applet mit einem *Label*, das sonst gar nichts tut, sieht so aus:

```
import java.awt.*;
import java.applet.*;

public class Applet1 extends Applet
{
    Label hallo = new Label("Hallo!");

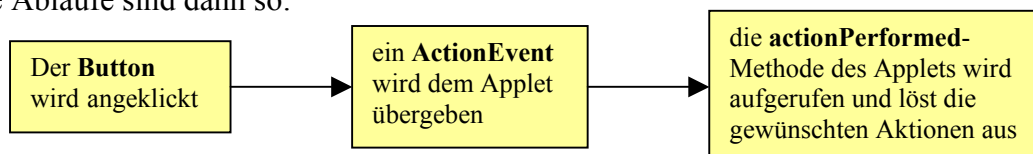
    public void init()
    {
        setLayout(null);
        hallo.setBounds(50, 30, 50, 20);
        add(hallo);
    }
}
```



## 2.2 Die Verarbeitung von Ereignissen

Wir wollen uns hier auf Reaktionen beschränken, die auf das Anklicken eines Buttons erfolgen. Dazu müssen wir den Button – als *Ereignissender* - mit einem *Ereignisempfänger* verbinden (einem *Listener*), der die gewünschten Aktionen mit einer *Ereignisbehandlungsmethode* auslöst. Wir wollen beim Button das *ActionEvent* benutzen, das durch das Anklicken ausgelöst wird. (Genauso gut hätten wir auch ein Mausereignis nehmen können!) Es gibt in Java sehr viele verschiedene Möglichkeiten, auf Ereignisse zu reagieren. Wir wollen hier ein sehr einfaches Verfahren wählen, bei dem *das Applet selbst* zum *Listener* gemacht wird, indem es das *ActionListener-Interface* implementiert. Dazu muss eine Methode *actionPerformed* geschrieben werden.

Die Abläufe sind dann so:



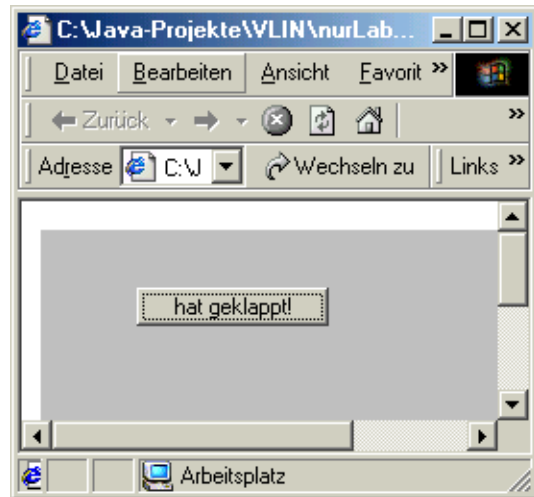
Ein Beispiel, bei dem sich auf Mausklick die Aufschrift des Buttons ändert, sieht so aus:

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

public class Applet1 extends Applet
    implements ActionListener
{
    Button hallo = new Button("Hallo!");

    public void init()
    {
        setLayout(null);
        hallo.setBounds(50,30,100,20);
        hallo.addActionListener(this);
        add(hallo);
    }

    public void actionPerformed(ActionEvent e)
    {hallo.setLabel("hat geklappt!");}
}
```



### 2.3 Ein Applet zur Ein- und Ausgabe von Zeichenketten

Wir wollen zwei Textfelder benutzen, deren Inhalte wird mit den Methoden *getText()* bzw. *setText()* lesen und beschreiben können. Ein Start-Button soll die Verarbeitung einleiten. Zwei Label-Komponenten beschriften die Felder. Der Inhalt des oberen Eingabefeldes wird in Grossbuchstaben im unteren dargestellt.

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

public class Applet1 extends Applet implements ActionListener
{
    Label lrein = new Label("Eingabe:");
    Label lraus = new Label("Ausgabe:");
    Button start = new Button("start!");
    TextField rein = new TextField(40);
    TextField raus = new TextField(40);
    String eingabe,ausgabe;

    public void init()
    {
        setLayout(null);
        lrein.setBounds(10,10,50,15); add(lrein);
        rein.setBounds(60,10,200,20); add(rein);
        lraus.setBounds(10,50,50,15); add(lraus);
        raus.setBounds(60,50,200,20); add(raus);
        start.addActionListener(this);
        start.setBounds(145,90,30,15); add(start);
    }
}
```



Das Applet als Action-Listener deklarieren.

GUI-Komponenten erzeugen

GUI darstel-

```

public void actionPerformed(ActionEvent e)
{
    eingabe = rein.getText();
    ausgabe = eingabe.toUpperCase();
    raus.setText(ausgabe);
}
}

```

Zeichenketten verarbeiten

### 3. Methoden zur Zeichenkettenverarbeitung

Zeichenketten bezeichnet man als Strings. Es gibt eine *String*-Klasse und eine *StringBuffer*-Klasse. Sie unterscheiden sich dadurch, dass Strings als final deklariert, StringBuffer aber dynamisch sind. Verändert man also Strings, so erhält man einen *neuen* String, während StringBuffer *geändert* werden. (Es gibt auch noch andere Unterschiede.) Für unsere Zwecke sind die Unterschiede ohne Bedeutung: wir verwenden Strings.

Einen Strings können wir uns als eine Folge von Zeichen vorstellen, wobei die Nummerierung bei Null beginnt.

Die Zählung beginnt bei 0!

D	a	s		i	s	t		e	i	n		S	t	r	i	n	g
---	---	---	--	---	---	---	--	---	---	---	--	---	---	---	---	---	---

Im String stehen 16-Bit-Unicode-Zeichen. Bei Bedarf können diese Zeichen durch *Typecasting* in ganze Zahlen umwandeln - z. B. um die Ordnungsnummer des Zeichens zu bestimmen: `int i = (int)'c';` Will man solche Zahlen in Textfeldern darstellen, dann gibt es drei „Tricks“ dafür:

- Entweder man verwandelt die ganze Zahl in ein Objekt der *Wrapper-Klasse Integer*, das dann in einen String umgewandelt wird:  
`raus.setText(new Integer(i).toString());`
- oder man hängt die Zahl einfach an einen existierenden String (z. B. ein Leerzeichen) an – dann erfolgt eine automatische Typumwandlung. Zur Not kann man das erste Zeichen dann wieder entfernen:  
`raus.setText(" "+i);`
- oder man benutzt eine der statischen Umwandlungsmethoden *valueOf()* der String-Klasse:  
`raus.setText(String.valueOf(i));`

Von den zahlreichen Methoden der String-Klasse wollen wir nur einige benutzen:

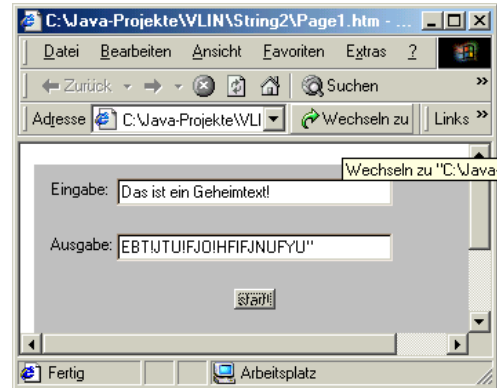
<b>charAt</b> (int)	liefert das entsprechende Zeichen zurück
<b>compareTo</b> (String)	vergleicht mit einem String
<b>concat</b> (String)	verbindet mit einem String
<b>indexOf</b> (String)	liefert den Index des ersten Zeichens des Teilstrings
<b>length</b> ()	liefert die Länge des Strings
<b>replace</b> (char1, char2)	ersetzt alle Zeichen char1 durch char2
<b>substring</b> (int, int)	liefert einen Teilstring
<b>toLowerCase</b> ()	konvertiert in Kleinschreibung
<b>toUpperCase</b> ()	konvertiert in Grossschreibung
<b>toString</b> ()	wandelt ein Objekt in einen String um

## 4. Verschlüsselungsverfahren

Eines der ältesten kryptografischen Verfahren ist das *Cäsar-Verfahren*, bei dem alle Zeichen zyklisch um eine Stelle im Alphabet verschoben werden:  $A \rightarrow B, B \rightarrow C, \dots, Z \rightarrow A$

In der folgenden Verschlüsselungsmethode wird ausgiebig vom Typecasting Gebrauch gemacht: Zeichen werden in Zahlen und diese wieder in Zeichen umgewandelt.

```
public void actionPerformed(ActionEvent e)
{
    char c,d;
    int nr,max,min;
    eingabe = rein.getText().toUpperCase();
    ausgabe = "";
    max = (int)'Z';
    min = (int)'A';
    for(int i=0;i<eingabe.length();i++)
    {
        c = eingabe.charAt(i);
        nr = (int)c;
        nr = nr + 1;
        if (nr > max) nr = (nr-max)+min-1;
        ausgabe = ausgabe+(char)nr;
    }
    raus.setText(ausgabe);
}
}
```



Die Zeichen können natürlich auch um andere Werte verschoben werden.

### *Versetzungsverfahren:*

Die Zeichen werden zeilenweise in einen „Block“ der Breite  $n$  geschrieben und spaltenweise wieder ausgelesen:

Klartext: *DIE GROESSTEN KRITIKER DER ELCHE WAREN FRUEHER SELBER WELCHE*

Block mit  $n=8$ :

D	I	E		G	R	O	E
S	S	T	E	N		K	R
I	T	I	K	E	R		D
E	R		E	L	C	H	E
W	A	R	E	N		F	R
U	E	H	E	R		S	E
L	B	E	R		W	E	L
C	H	E					

Geheimtext: *DSIEWULCISTRAEBHETI RHEE EKEEER GNELNR R RC W OK HFSE ERDEREL*

### *XOR-Verschlüsselung:*

Der Klartext wird mit Hilfe eines Schlüsselwortes verschlüsselt. Dazu werden Klartext und der ggf. wiederholte Schlüssel untereinander geschrieben. Die Ordnungsnummern der Zeichen werden in die Dualdarstellung gebracht und bitweise xor-verschlüsselt: Stehen zwei gleiche Ziffern (Nullen oder Einsen) übereinander, dann ergibt sich eine Null, sonst eine Eins. Das Verfahren ist durch nochmalige Verschlüsselung umkehrbar und sehr schnell. Die Schlüssel können z. B. in einer Chipkarte gespeichert sein.



Wir wollen das Verfahren an einem Beispiel durchgehen:

Als Primzahlen wählen wir:  $p = 97$  und  $q = 151$ ,  
 deren Produkt ist  $n = 14647$ .  
 Als  $r$  wählen wir beliebig  $r = 11$ ,  
 und damit das Produkt  $x*y = 96*150*11+1 = 144001$ .  
 $x$  bestimmen wir so,  
 dass auch  $y$  eine ganze Zahl ist  $x = 19$   
 und erhalten für  $y$   $y = 7579$ .

Jetzt kann es losgehen:

Wir verschlüsseln das Klartextzeichen „A“:  
 Ordnungsnummer  $Z$  von „A“:  $Z = 65$   
 verschlüsselte Zeichennummer  $V$ :  $V = 65^{19} \text{ modulo } 14647 = 12561$

Der Empfänger möchte wieder entschlüsseln:  
 Geheimzeichennummer  $V$ :  $V = 12561$   
 Klartextzeichennummer  $Z$ :  $Z = 12561^{7579} \text{ modulo } 14647 = 65$   
 Klartextzeichen: „A“

Da sich bei dem Verfahren riesige Zahlen durch die Potenzen ergeben, benötigen wir noch ein schnelles Verfahren, um die Reste dieser Potenzen zu berechnen:

**Berechnung von  $r = \text{Basis}^{\text{Exponent}} \text{ modulo } n$ :**

$b \leftarrow$ Basis	
$e \leftarrow$ Exponent	
$r \leftarrow 1$	
SOLANGE $e \neq 0$ TUE	
$e$ ist eine gerade Zahl	
Wahr	Falsch
$b \leftarrow (b*b) \text{ MOD } n$	$e \leftarrow e -$
$e \leftarrow e \text{ DIV } 2$	$r \leftarrow (r * b) \text{ MOD}$

## 5. Aufgaben:

1. a: Realisieren Sie das Cäsar-Verfahren für beliebige Verschiebungswerte. Entwickeln Sie eine GUI, in der die Verschiebungsweite ausgewählt werden kann. Entschlüsseln Sie die Texte auch wieder.  
b: Realisieren Sie das Versetzungsverfahren. Ermöglichen Sie auch hier Wahlmöglichkeiten.  
c: Realisieren Sie die XOR-Verschlüsselung. Wandeln Sie dazu die Zeichen in Bitfolgen um. Probieren Sie alternativ den XOR-Operator für Zahlen aus.
2. a: Realisieren Sie das beschriebene Verfahren zur Berechnung von Resten großer Potenzen.  
b: Benutzen Sie alternativ für diese Berechnung die Java-Klasse *BigInteger*.
3. Verschlüsseln Sie längere Texte und stellen Sie die Verteilung der Zeichen in Tabellenform oder grafisch als Histogramm dar. Versuchen Sie die Zeichen anhand der Zeichenhäufigkeit zu entschlüsseln. Funktioniert das bei allen Verfahren?
4. a: Extrahieren Sie eine Zahl, die als Zeichenkette eingegeben wurde. Benutzen Sie das Java-Hilfesystem.  
b: Geben Sie Rechenaufgaben als Zeichenketten ein: „3+4=“, „12-5=“, ... Suchen Sie die Zahlen und die Operatoren im String und geben Sie das Ergebnis aus.  
c: Analysieren Sie Polynome, die als String eingegeben werden. Beginnen Sie mit einstelligen ganzen Zahlen als Koeffizienten und Exponenten. „ $f(x)=3*x^2-2*x-1$ “. Weiten Sie dann die Zahlendarstellung aus.  
d: Stellen Sie die Graphen der eingegebenen Funktionen dar.
5. a: Benutzen Sie die Lösung aus 4.b, um die Ableitung ganzrationaler Funktionen zu bestimmen.  
b: Diskutieren Sie das Problem, Produkt-, Quotienten- und Kettenregel auf entsprechende Funktionen anzuwenden. Welche Teilprobleme müssen gelöst werden?