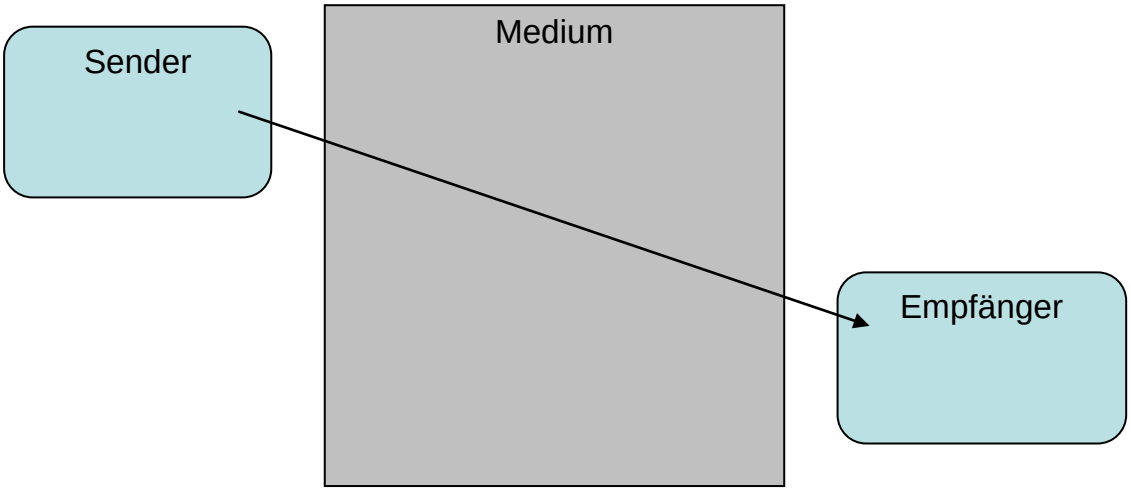
28 Kryptografie 1

28.0 Kodieren, Verschleiern, Authentifizieren

Der Weg der Nachricht: „Ich komme heute zum Abendessen.“



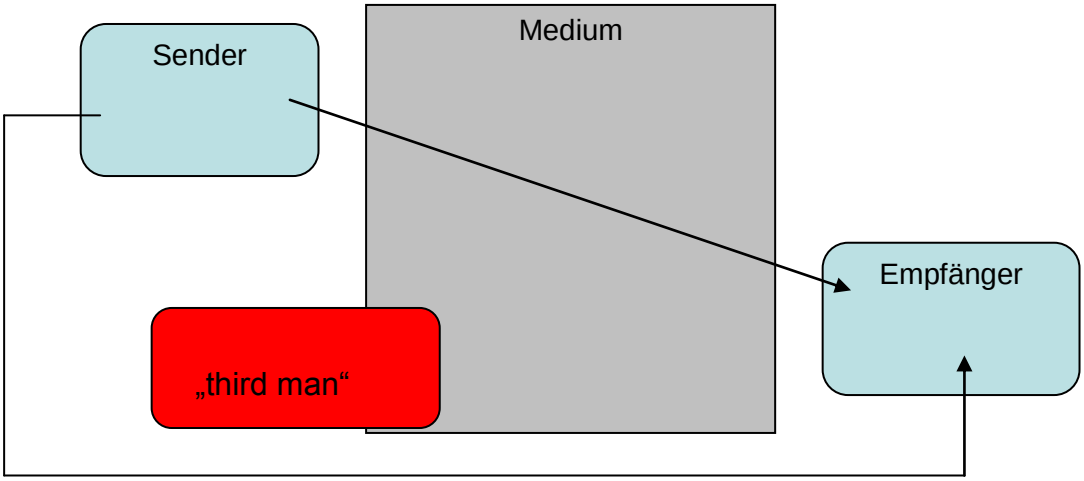
Bei einem Wechsel des Mediums muss die Nachricht umgesetzt werden. Dies erfolgt nach festgelegten Vorschriften. Schon die Übertragung aus der gesprochenen Sprache in die Schriftsprache ist eine solche Umsetzung. Allgemein spricht man von Kodieren.
siehe dazu auch: Morsecode, ASCII – „Code“, Unicode, Codex, Encoder, Decoder.

> Ein **Code** (engl. von lat. codex Gesetzbuch) ist im weitesten Sinn eine *Vorschrift*, etwa für eine solche Umsetzung, aber auch für die Zuordnung einer Abkürzung, eines Schlüssels, bis hin zu Konventionen und Verhaltensregeln. In der deutschen Sprache versteht man darunter oft eine geheim gehaltene Vorschrift zur Verschlüsselung (Verschleierung) von Nachrichten.

Die theoretische Informatik liefert dazu den Begriff des „Übersetzers“ und „Compilers“ (s. Kap. „Endliche Automaten“).

Bei der Arbeit am Computer finden solche Übersetzungen immer wieder statt: Diesen Satz hier habe ich zuerst gesprochen, dann geschrieben, auf der Tastatur getippt. Die im Rechner angekommenen Tastaturcodes wurden als Zeichen auf einem deutschen Tastaturlayout gedeutet und entsprechend in ANSI-Code oder Unicode umgesetzt. Beim Ausdruck hat der Druckprozessor das „Bild“ zu den einzelnen Buchstaben produziert und als Punktraster ausgedruckt. Bei der Speicherung auf Festplatte wird die Information natürlich noch in magnetische Zeichen umcodiert …
Bei der digitalen Speicherung von Musik und Film werden diese Umwandlungen öfter zum Problem, weil es mehrere konkurrierende Verfahren gibt (etwa WMA, MP3, AAC bei Musik) und Programme nicht alle Codes verstehen oder verstehen wollen oder die Codecs urheberrechtlich geschützt sind..

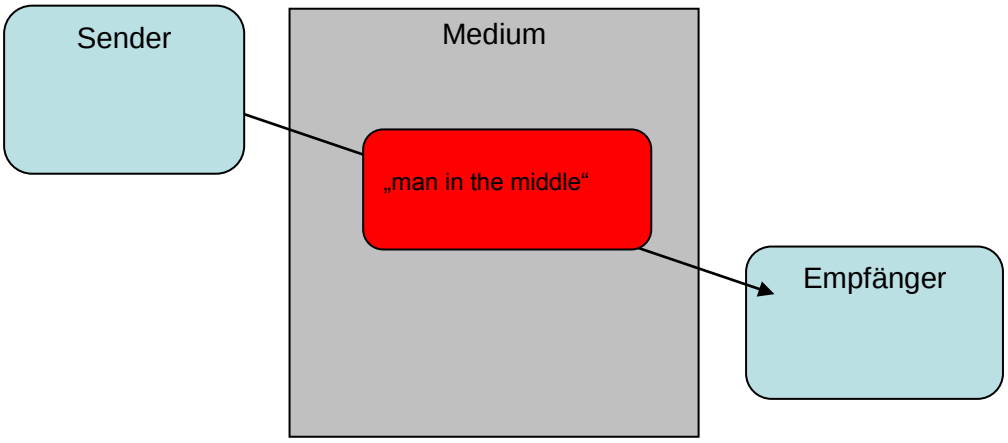
Bei der Verschleierung von Informationen (fast immer von Informationen in Textform) ist die Situation ähnlich:



Der Schlüssel nimmt einen „Umweg“

Hier kommt ein Dritter, ein Außenstehender hinzu, der evtl. die Nachrichten in dem Medium mithört (den Boten abfängt, den Funkspruch abhört, …)
Damit dieser Außenstehende die Information nicht erhält, muss ein **geheimer Schlüssel** (=Code) her. Doch wie kommt der rechtmäßige Empfänger an diesen Schlüssel? Besser nicht auf demselben Weg wie die Nachricht!

Sehen wir uns nun die Situation noch aus der Sicht des Empfängers an. Er bekommt eine Nachricht. Aber stammt diese auch vom Sender, ist sie unverändert? Ein „man in the middle“ könnte ja bei einem Überweisungsauftrag, den ich meiner Bank schicke, den Zahlungsempfänger fälschen.
Eng verbunden mit Verschlüsselung von Nachrichten sind also die Problemfelder Authentifizierung und Sicherung der Integrität der Nachricht.



Weshalb beschäftigen wir uns heute mit diesen Problemen, die früher ja weitgehend dem Militär überlassen blieben?

Das Internet ist von seiner Struktur her ein quasi öffentliches Medium. Datenpakete werden fast nie direkt vom Sender zum Empfänger geschickt, sondern immer über eine nicht vorhersehbare Reihe von Zwischenstationen. Emails sind deshalb mit Postkarten vergleichbar, die ja auch gelesen werden können, ohne dass Sender oder Empfänger dies merken. Verschlüsselung schafft dann so eine Art „Briefumschlag“, die Verwendung von „Public-Key“-Systemen kann Stempel und Unterschrift ersetzen.

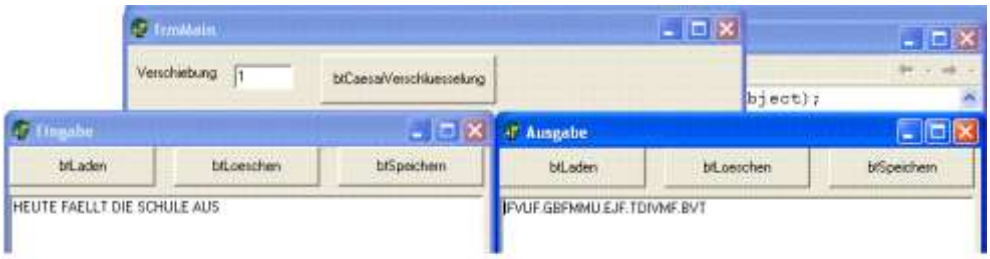
Von **symmetrischen Verfahren** spricht man, wenn die Dekodierung mit demselben Verfahren (demselben Schlüssel) erfolgt wie die Verschlüsselung.
Bei **asymmetischen Verfahren** werden für Ver- und Entschlüsselung grundsätzlich unterschiedliche Schlüssel gebraucht (s.u.; RSA).

Grundlegende symmetrische Verfahren sind Substitution und Transposition

Bei der **Substitution** werden die einzelnen Zeichen durch andere ersetzt.
Einfachste Möglichkeit: Man ersetzt jeden Buchstaben durch den übernächsten im Alphabet, also A durch C, B durch D, C durch E usw. (Cäsar – Verfahren; s.u.).
Allgemein bildet man eine beliebige Vertauschung (Permutation) der Zeichen des Alphabets.
Verwendet man für alle Zeichen eines längeren Textes denselben Schlüssel, ist das Verfahren statistisch angreifbar. Abhilfe schafft, aufeinander folgende Zeichen jeweils anders zu verschlüsseln (s. etwa Vigenère – Verfahren). Die Länge des Schlüssels spielt für die Sicherheit des Verfahrens eine ganz wichtige Rolle. Extrem und (kryptografisch) sicher ist der „One time pad“; hierbei ist der Schlüssel genauso lang wie die Nachricht und wird nur einmal benutzt. Aber wie kommt dieser Schlüssel zum Empfänger ?

Bei einer **Transposition** ändert man die Reihenfolge der Zeichen.
Im einfachsten Fall schreibt man jede Zeile von hinten nach vorne auf.
Oder man trägt die Buchstaben zeilenweise in eine Tabelle fester Größe ein, um sie dann spaltenweise auszulesen.

Aktuelle, zur Zeit sichere symmetrische Verschlüsselungsverfahren wie **AES** (Advanced Encryption Standard) benutzen eine geschickte Kombination von Substitutionen und Transpositionen, um die Nachteile der einzelnen Verfahren auszugleichen.

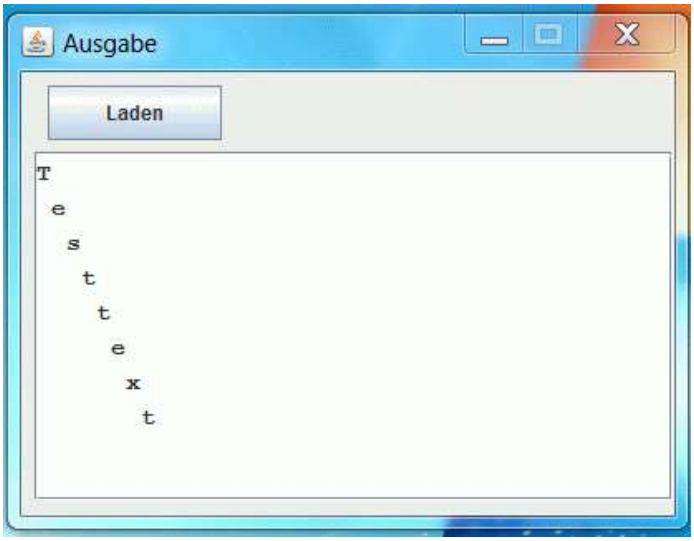


Wie schreibt man nun ein Programm zur Verschlüsselung?

Wir verschlüsseln Texte. Texte haben eine Zeilenstruktur.
Jede Zeile ist ein String und besteht damit aus einer Reihe von Zeichen (Datentyp char).
Es erscheint sinnvoll, zunächst den Umgang mit Texten, den dabei auftretenden Datentypen und den dabei benutzten algorithmischen Strukturen (i. w. Schleifen) zu üben, bevor die Verfahren zur Verschlüsselung programmiert werden.

28.1 Ein universelles Ein - Ausgabe – Formular

Als Basis für die weiteren Übungen erstellen wir ein Formular zur Ein- und Ausgabe von Texten. Objektorientierter Nebeneffekt: Es ist eines der eher seltenen Beispiele, wo sinnvoll mehrere Formulare derselben Klasse nebeneinander verwendet werden.



Das Formular besteht im Wesentlichen aus einem Textarea – Feld. Es kann einen Text in das Feld laden und seinen Inhalt speichern; beides geschieht mit Unterstützung durch die entsprechende Swing– Dialogkomponente.

Die Formularklasse stellt folgende Methoden für Aufträge und Anfragen zur Verfügung:

int getAnzahl ()	ermittelt die Anzahl der Textzeilen
String getZeile (int nr)	liefert die angegebene Zeile als String
void loeschen ()	löscht den Inhalt des Textarea
void schreibeZeile (String s)	fügt den String als neue Zeile an den Text an

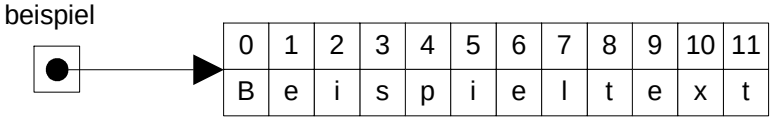
Das Ein - Ausgabe – Formular ist besonders praktisch, wenn bei einem Verschlüsselungsprogramm Quelltext und verschlüsselter Text gleichzeitig betrachtet und unabhängig von einander geladen und gespeichert werden sollen. Es kann aber auch in den nächsten Projekten benutzt werden.

28.2 Stringmethoden

Die Klasse *String*

Objekte der Klasse **`String`** können Zeichenketten verwalten.

Ein String-Objekt erzeugen
Ein Beispielobjekt definieren: **`String beispiel;`**
Eine Zeichenkette zuweisen: **`beispiel = "Beispieltext";`**

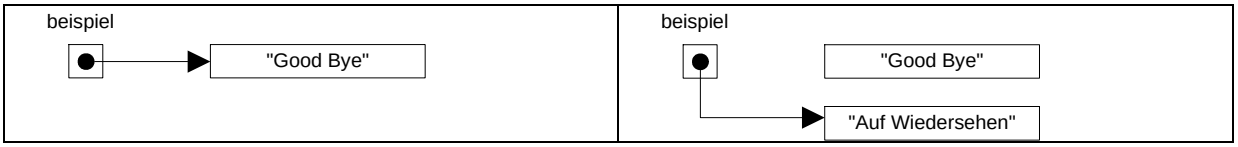


Die einzelnen Zeichen der Zeichenkette sind vom Datentyp **`char`**, die Nummerierung beginnt mit dem Index **`0`**. Die Zeichenkette **`beispiel`** hat die Länge **`12`**.

Einem String-Objekt eine neue Zeichenkette zuweisen

```
String beispiel;
```

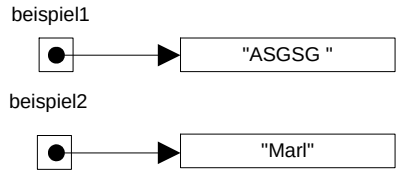
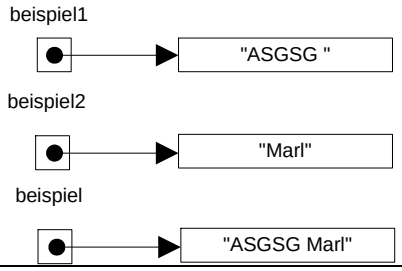
`beispiel = "Good bye";`	`beispiel = "Auf Wiedersehen";`



Ein neues String-Objekt wird erzeugt. Das alte String-Objekt ist nicht mehr referenziert.

String-Objekte verketten

String **`beispiel1, beispiel2, beispiel;`**

`beispiel1 = "ASGSG ";` **`beispiel2 = "Marl";`**	**`beispiel = beispiel1 + beispiel2;`**
	

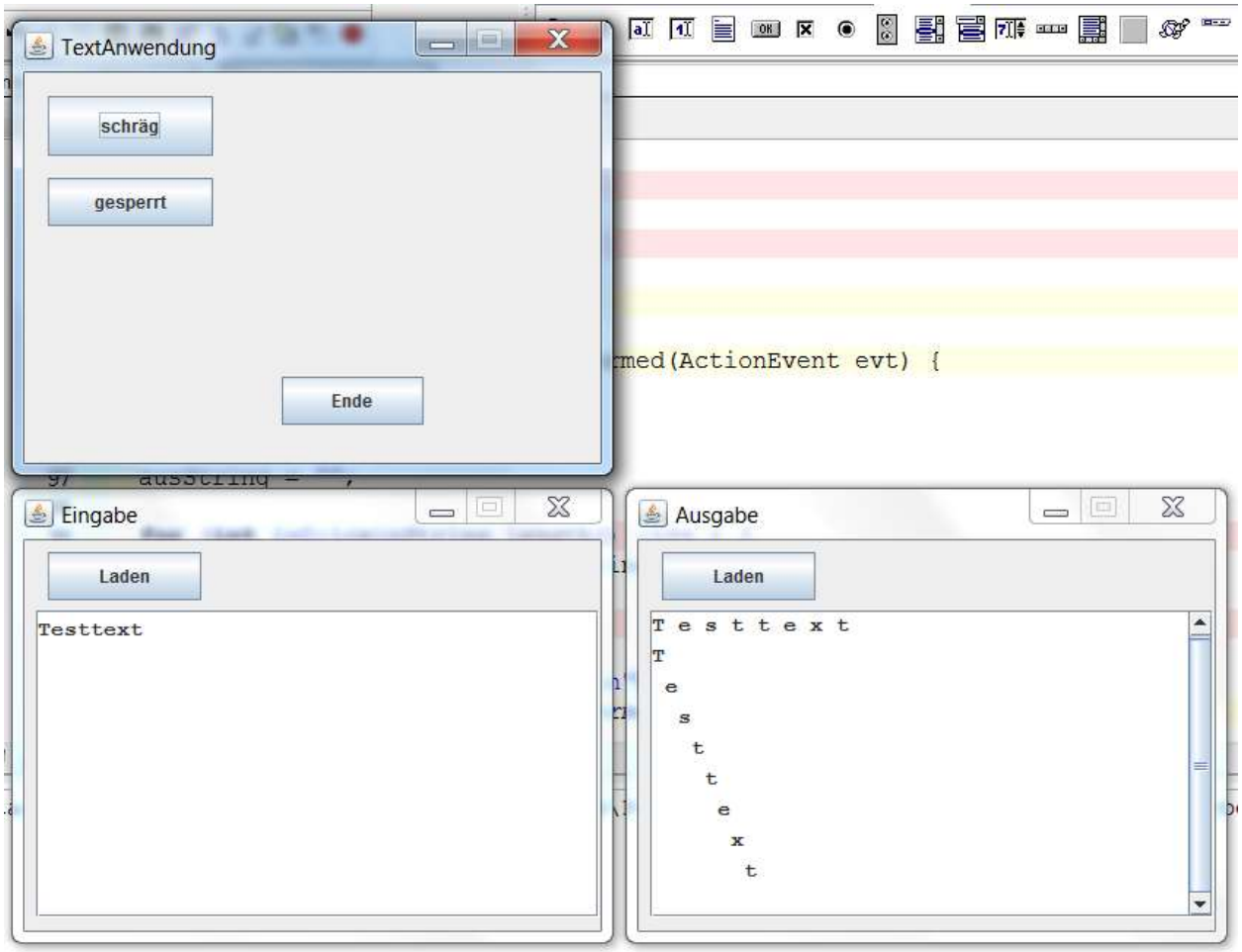
Methoden der Klasse *String*

Anfrage	**`int length()`**
	Liefert die Länge der Zeichenkette.
Anfrage	**`int indexOf(String pStr)`**
	Liefert die erste Position, an der **`pStr`** in dieser Zeichenkette vorkommt. Steht **`pStr`** direkt am Anfang dieser Zeichenkette, wird 0 zurückgeliefert. Ist **`pStr`** gar nicht enthalten, wird -1 geliefert.
Anfrage	**`String substring(int pIndexAnfang)`**
	Liefert einen neuen String, der nur die Zeichen ab der Position **`pIndexAnfang`** bis zum Ende dieser Zeichenkette enthält.
Anfrage	**`String substring (int pIndexAnfang, int pIndexEnde)`**
	Liefert einen neuen String, der nur die Zeichen ab der Position **`pIndexAnfang`** bis zur Position **`pIndexEnde`** enthält. **`pIndexEnde`** bezeichnet hierbei den Index hinter dem letzten Zeichen, das kopiert werden soll.
Anfrage	**`char charAt(int pIndex)`**
	Liefert das Zeichen, das an der Position **`pIndex`** steht. Das erste Zeichen wird mit dem Wert 0 ausgelesen.
Anfrage	**`boolean equals(Object pObject)`**
	Liefert genau dann **`true`**, wenn **`pObject`** ein Exemplar der Klasse **`String`** ist, das denselben Wert besitzt wie dieses Objekt, sonst **`false`**.
Anfrage	**`int compareTo(String pString)`**
	Liefert einen Wert, der kleiner als 0 ist, wenn die Zeichenkette lexikalisch kleiner ist als **`pString`**, liefert 0, wenn sie gleich **`pString`** ist, und einen Wert größer als 0, wenn die Zeichenkette lexikalisch größer ist als **`pString`**.
Anfrage	**`String replace(String pAlt, String pNeu)`**
	Liefert einen neuen String, die dem jede Zeichenfolge **`pAlt`** durch die Zeichenfolge **`pNeu`** ersetzt worden ist.

Anfrage	**`String toLowerCase()`**
	Liefert einen neuen String, die dem jeder Großbuchstabe durch den entsprechenden Kleinbuchstaben ersetzt worden ist.
Anfrage	**`String toUpperCase()`**
	Liefert einen neuen String, die dem jeder Kleinbuchstabe durch den entsprechenden Großbuchstaben ersetzt worden ist.
Anfrage	**`String concat(String pString)`**
	Liefert einen neuen String, der die Verkettung dieses Strings mit **`pString`** enthält.

28.3 Zerlegen und Zusammenbauen von Texten

Im Projekt „Texte bearbeiten“ verwenden wir zwei Ein- Ausgabe – Formulare. Die erste Zeile im linken Formular wird rechts „umgearbeitet“ dargestellt.



Beim „gesperrt – Ausgeben“ wird hinter jedem Zeichen ein Leerzeichen eingefügt.

mögliche Lösung:

```
public void btGesperrt_ActionPerformed(ActionEvent evt) {
    String einString, ausString;
```

```
    einString = ein.getZeile(0);
    ausString = "";

    for (int i=0;i<einString.length() ;i++ ) {

      ausString = ausString + einString.charAt(i) +" ";

    } // end of for

    aus.schreibeZeile(ausString + "\n");
  } // end of btGesperrt_ActionPerformed
```

Mit der For – Schleife gehen wir Zeichen für Zeichen des eingegebenen Textes durch und hängen den jeweiligen Buchstaben **`einString.charAt(i)`** an den zu erstellenden String an, gefolgt von einem Leerzeichen.

Für den Schrägdruck verwenden wir eine private Hilfsmethode (des Hauptformulars), die eine anzugebende Zahl von Leerzeichen zurückgibt:

```
public String leer(int anzahl) {
    String s = "";
    for (int i=0; i<anzahl ;i++ ) {
      s = s + " ";
    } // end of for
    return s;
  }
```

Die Implementation der „btSchraeg“ – ActionPerformed– Methode folgt im Aufbau dem Gesperrt – Ausgeben; das „schreibeZeile“ steht jetzt aber innerhalb der Schleife, so dass für jeden Buchstaben eine Zeile produziert wird.

28.4 Aufgaben

a) Die Buchstaben eines Wortes verdoppeln : HHaallllloo

b) Die Buchstaben eines Wortes untereinander ausgeben

c) Die Buchstaben eines Wortes rückwärts ausgeben

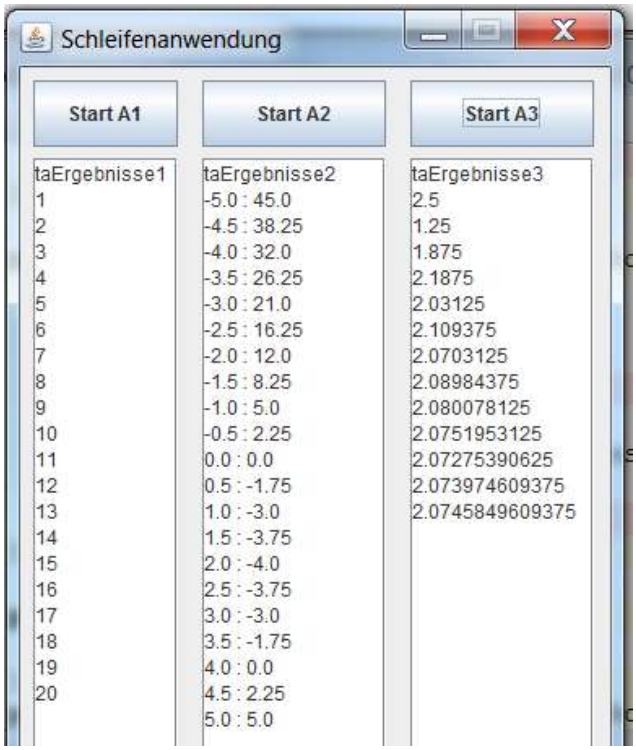
d) Feststellen, ob es sich bei einem Wort um ein Palindrom handelt (anna, otto, lagerregal ...)

e) Worte abbauen und wieder aufbauen: Test Tes Te T T Te Tes Test

f) Mit Buchstaben „malen“, ausgehend von btSchraeg.

g) Die Buchstaben in einem String sortieren.

28.5 Iterationsübungen (Aufgaben mit Schleifen)



Zuerst einige Beispiele:

1) alle Zahle von 1 bis 20 anzeigen

```
public void btStart1_ActionPerformed
                     (ActionEvent evt) {
    for (int i = 1; i <=20; i++) {
      taErgebnisse1.setText (
       taErgebnisse1.getText()+ "\n" + i);

    } // end of for
  }
```

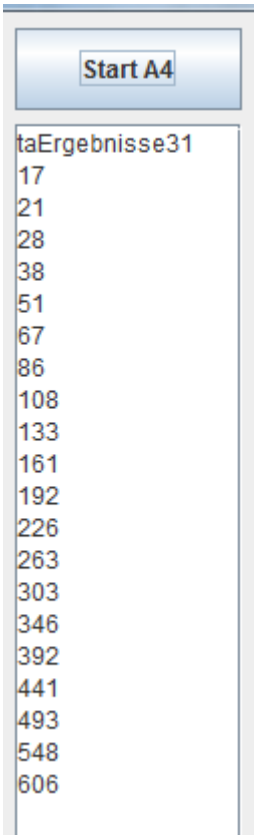
2) Wertetabelle einer Funktion f von –5 bis 5

```
public void btStart2_ActionPerformed(ActionEvent evt) {
    double x, fx;

    x = -5;
    while (x <= 5) {
      fx = x * x - 4 * x;
      taErgebnisse2.setText ( taErgebnisse2.getText()+ "\n" + x + " : " + fx);
      x = x + 0.5;
    } // end of while
  } //
```

3) Nullstellensuche im Bereich von 0 bis 5 bei einer Funktion f (Bisektion)

```
private double f ( double x) {                  // Funktion f
    return x*x*x*x - 7*x -4;
  }

public void btStart3_ActionPerformed(ActionEvent evt) {
    double links, rechts, mitte;

    links = 0;  rechts = 5;
    //f muss an den Intervallgrenzen unterschiedliches Vorzeichen haben

    while (rechts - links > 0.001) {    //3 Nachkommastellen Genauigkeit
      mitte = 0.5 *(links + rechts);

      if (f(links) * f(mitte) < 0)      // unterschiedliche Vorzeichen
      rechts = mitte;
      else links = mitte;

      taErgebnisse3.setText ( taErgebnisse3.getText()+ "\n" + mitte);

    } // end of while
}
```

4) eine Zahlenfolge



Bei einfachen Beispielen wie etwa:
2, 6, 10, 14, 18, 22, …
ist die Differenz benachbarter Zahlen gleich.
In einer Schleife wird dann der Wert jeweils um die Differenz erhöht:

```
zahl = 2;
for (int i = 1; i <= 20; i++) {
   taErgebnisse4.setText ( taErgebnisse4.getText()+ "\n" + i);
   zahl = zahl + 4;
}
```

Das Beispiel nebenan ist etwas anspruchsvoller. Die Differenzen bilden eine aufsteigende Folge (4, 7, 10, 13, 16, …); die Differenzen zweiter Ordnung sind dann konstant (=3).

Programmierung:
Anstelle der Konstanten „4“ aus dem ersten Beispiel brauchen wir eine zweite Variable Differenz, die jeweils um 3 erhöht wird.

28.6 Aufgaben mit Schleifen

a) alle Quadratzahlen von 1 bis 400

b) alle Kubikzahlen von 1 bis 27000

c) die Wertetabelle von $f(x) = x^4 - 7x - 4$ im Bereich von 0 bis 5 mit Zuwachs 0,5 (0,1).

d) die Reihe 3, 8, 15, 24, 35, 48, 63, ...

e) die Reihe 27, 23, 37, 33, 47, 43, 57, 53, ...

f) Nullstellensuche bei $f(x) = x^5 - 17x^2 + 3$ (geeignetes Intervall?)

g) Fibonacci – Folge: 1 1 2 3 5 8 13 21 34 .... (jede Zahl ist Summe der beiden Vorgänger)

28.7 Verschlüsselung nach Caesar

(nach Damann/Wemßen, Kap.23)

Schon Julius Caesar benutzte ein nach ihm benanntes Verfahren, um wichtige Nachrichten „unlesbar“ für Dritte an den Empfänger zu übermitteln. Bei Sueton lesen wir über den römischen Staatsmann Caesar (100-44 v.Chr.):

> „Es existieren auch [Briefe von Caesar] an Cicero und an Bekannte über Dinge, in denen er, wenn etwas vertraulich übermittelt werden musste, in Geheimschrift schrieb, d. h. er veränderte die Ordnung der Buchstaben derart, dass kein einziges Wort mehr ausgemacht werden konnte. Wenn jemand das entziffern und den Inhalt erkennen wollte, so musste er den vierten Buchstaben des Alphabets, also D, für A einsetzen, und so mit den anderen.“

Diesem Verschlüsselungsverfahren liegt ein einfaches Schema zugrunde: Die Buchstaben des Alphabets werden um einen oder mehrere Buchstaben verschoben:

Alphabet:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Schlüssel:	B C D E F G H I J K L M N O P Q R S T U V W X Y Z A

Klartext: **HEUTE GIBT ES HITZEFREI!**

Chiffre: **IFVUF HJCU FT IJUAFGSFJ!**

Für die folgenden Überlegungen legen wir fest:

- Alle deutschen Umlaute werden ersetzt durch „ae“, „oe“, „ue“ und „ß“ durch „ss“.
- Alle Buchstaben werden in Großbuchstaben umgewandelt.
- Leerzeichen, Satzzeichen, Ziffern und Zeilenendemarken werden nicht verschlüsselt, im Gegensatz zu „echten“ Verschlüsselungen aber auch nicht herausgefiltert.

Alphabete

1. Möglichkeit: Wir geben ein Alphabet als String an in der Form

```
String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

Die Verschlüsselung erfolgt dann in einigen wenigen Schritten

```
{Position des unverschlüsselten Zeichens im Alphabet}
nr = alphabet.indexOf( einString.substring(i, i+1));
{Verschiebung der Position des Zeichens um IntSchluessel}
nr = (nr + verschiebung) % alphabet.length();
{und Ermittlung des verschlüsselten Zeichens}
ausString = ausString + alphabet.charAt(nr);
```

Zeichen, die nicht im Alphabet vorkommen, müssen über `if (nr > -1)` herausgefiltert und extra behandelt werden.

Anstelle des Alphabets nur aus Großbuchstaben kann man zusätzlich die kleinen Buchstaben aufnehmen, evtl. auch Ziffern und Satzzeichen.

2. Möglichkeit: Verschlüsselung mithilfe der ASCII-Tabelle

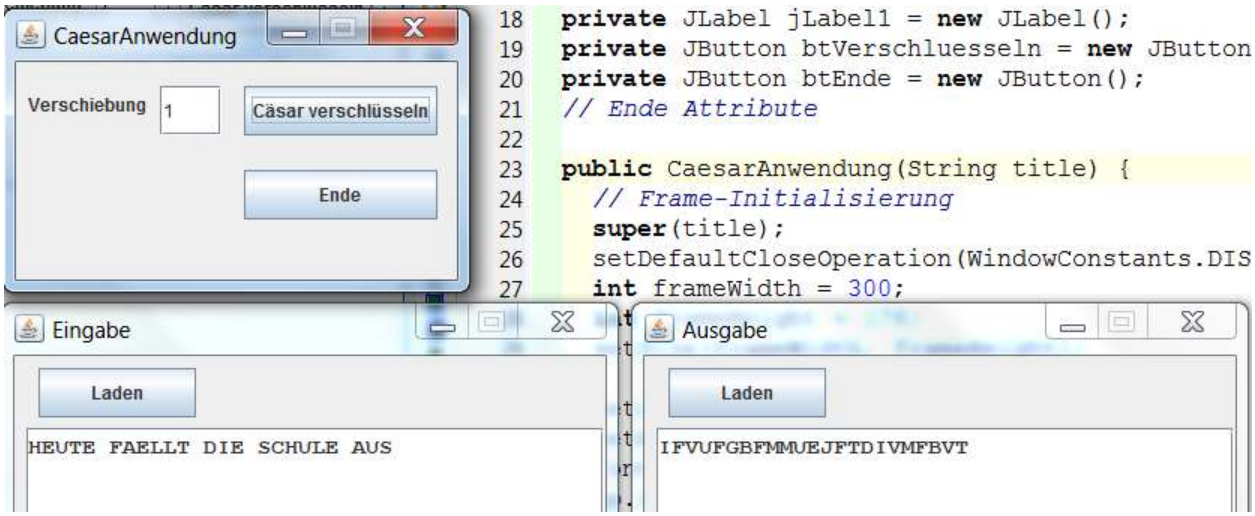
Mit anfang = (int) ‘A‘; ende = (int) ‘Z‘; (Hochkomma, weil char und kein String) errechnet man die Ordnungszahl des verschlüsselten Zeichens durch

```
{Position des unverschlüsselten Zeichens im Alphabet}
nr = (int) einString.charAt(i) - anfang;
{Verschiebung der Position des Zeichens um IntSchluessel
(= Schlüssel), Berechnung der Ordnungszahl in der ASCII-Tabelle
und Ermittlung des verschlüsselten Zeichens}
caesarNr := (nr + verschiebung) % 26;
caesarZeichen = (char) (caesarNr + anfang)
```

Man kann dann aber auch gleich alle ANSI (ASCII) – Zeichen als Alphabet betrachten:

```
caesarZeichen := (char)((int)einString.charAt(i)+verschiebung) % 256);
```

Aufgaben



a) Schreiben Sie das Programm **CaesarAnwendung1**, das einen vorgegebenen Klartext durch eine Verschiebung um n Zeichen (1 <= n < 26) verschlüsselt (Alphabet aus Großbuchstaben).

b) Schreiben Sie das Programm **CaesarAnwendung2**, welches zusätzlich einen durch eine Verschiebung verschlüsselten Text (Caesar-Verschlüsselung) wieder entschlüsselt (alle ASCII-Zeichen).

28.8 andere monoalphabetische Verschlüsselungen

Ein nach dem Caesar - Verfahren verschlüsselter Text kann bereits dann vollständig entschlüsselt werden, wenn nur für 1 Zeichen bekannt ist, wie es verschlüsselt wird.

Ein „sichereres“ Verfahren ordnet das Alphabet einem beliebigen, „willkürlich“ gewählten 2. Alphabet zu (mathematisch: Permutation der Zeichen). Der Empfänger der Nachricht muss die Verschlüsselungstabelle kennen, um den Text entschlüsseln zu können.

Klartext: `Dieser Text wird verschluesselt.`

Alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Schlüssel: E X D B F G Y U L K M J N O T Q R S A I P W C H Z V

Chiffre: `BLFAFS IFHI CLSB WFSADUJPFAAFJI.`

Programm **MonoAlphAnwendung (Verschlüsseln durch beliebiges Vertauschen)**

Schreiben Sie ein Programm **MonoAlphAnwendung** , das einen beliebigen Klartext mit einem beliebigen Alphabet (26 Buchstaben) verschlüsselt.

Variation

Um die Übertragung des Schlüssels an den Empfänger der verschlüsselten Nachricht zu vereinfachen, kann man auch eine Mischform der bisherigen Schlüssel bilden. Man vereinbart einen Schlüssel in Form eines Schlüsselwortes, hier als Beispiel das Schlüsselwort „Delphi“ sowie eine Zahl ≤ 26, hier z. B. die Zahl 10. Das Schlüsselwort wird im Schlüssel ab der 10. Stelle eingetragen, dann wird der Schlüssel durch die restlichen Buchstaben, reduziert um die Zeichen des Schlüsselwortes, durch Verschieben aufgefüllt.

Klartext: `Dieser Text wird verschluesselt.`

Schlüsselwort: **D E L P H I**
Schlüsselzahl: `10`

Alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
Schlüssel:	R	S	T	U	V	W	X	Y	Z	**D**	**E**	**L**	**P**	**H**	**I**	A	B	C	F	G	J	K	M	N	O	Q	

Chiffre: `UZVFVC GVNG MZCU KVCFTYLJVFFVLG.`

Aufgabe

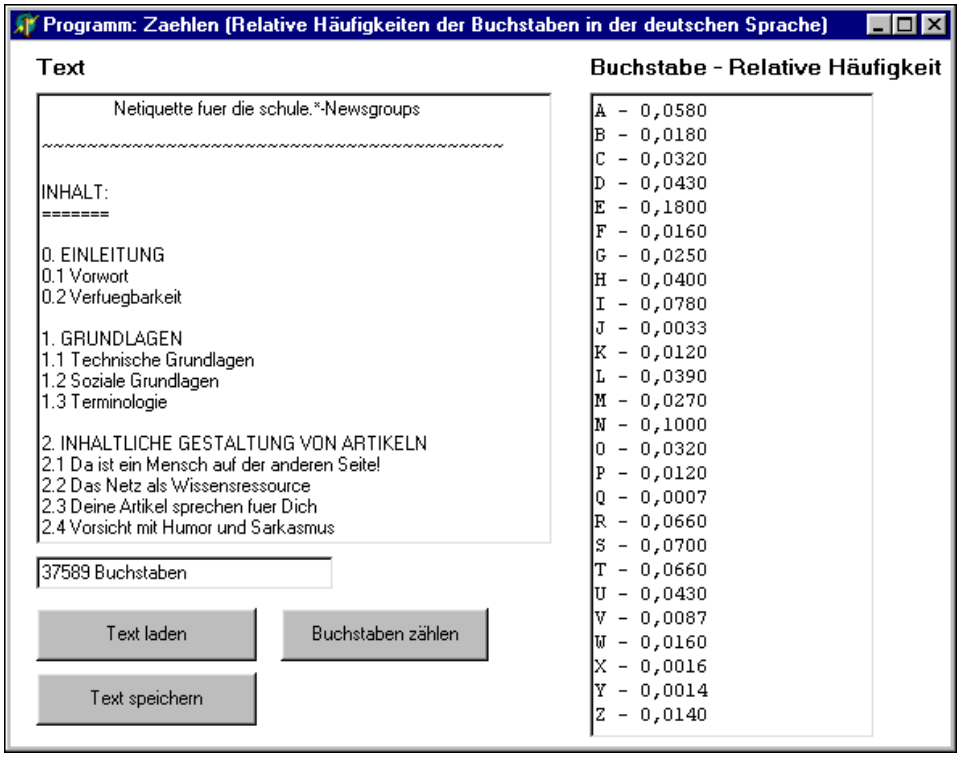
Erstellen Sie ein geeignetes Programm.

28.9 Ermitteln des Schlüssels

Um solche Geheimtexte zu „knacken“, kann man im Geheimtext die Häufigkeit der Buchstaben zählen und daraus Rückschlüsse auf die Kodierung ziehen. Durch Auszählen langer und verschiedenartiger Text erhält man eine Übersicht über die Wahrscheinlichkeiten für das Auftreten der einzelnen Buchstaben in der deutschen Sprache:

Buchstabe:	Wahrscheinlichkeit:	Buchstabe:	Wahrscheinlichkeit:
A	0,0433	P	0,0050
B	0,0160	Q	0,0001
C	0,0267	R	0,0686
D	0,0439	S	0,0539
E	0,1470	T	0,0473
F	0,0136	U	0,0319
G	0,0267	V	0,0074
H	0,0436	W	0,0142
I	0,0638	X	0,0001
J	0,0016	Y	0,0002
K	0,0096	Z	0,0142
L	0,0293	Ä	0,0049
M	0,0213	Ö	0,0025
N	0,0884	Ü	0,0058
O	0,0136	Leerzeichen	0,1515

Überprüfen Sie die Werte der Tabelle mit einem Programm **ZaehlenAnwendung**, mit dem Sie einen längeren Text (mindestens 1000 Buchstaben) in eine Textarea-Komponente einlesen und analysieren können. Lassen Sie die Buchstaben sowie die relativen Häufigkeiten, mit denen sie im Text auftreten, in einer zweiten Memokomponente ausgeben.



(Abb. Relative Häufigkeit der Buchstaben A .. Z in einem deutschen Text)

Aufgabe

Schreiben Sie ein Programm **Knacken**, das für einen verschlüsselten (längeren) Text eine entsprechende Tabelle erzeugt. Versuchen Sie mithilfe dieses Programms den verschlüsselten Text zu entschlüsseln, indem Sie

a) vom Programm eine Entschlüsselung durchführen lassen gemäß den ermittelten relativen Häufigkeiten der Buchstaben im chiffrierten Text,

b) vom Programm ein von Ihnen vorgegebenes Schlüsselzeichen gegen das vermutete „Originalzeichen“ austauschen lassen.

Die hier angesprochenen Verschlüsselungsverfahren gehören zu den einfachen, **monoalphabetischen Verschlüsselungen**. Auch für diese Verschlüsselungstechnik gibt es weitere interessante Varianten (z. B. Chiffrierung durch Multiplikation statt durch Addition).