Caesarchiffrierung mit JAVA

Bei der Caesarverschlüsselung hat man ein sehr einfaches Verfahren, bei dem aus dem Buchstaben des Originals durch eine einfache Berechnung der zugehörige verschlüsselte Buchstabe berechnet werden kann.
Ein Problem dabei ist aber, dass JAVA eine streng typgebundene Sprache ist, so dass nicht einfach mit Buchstaben gerechnet werden kann, sondern ein Buchstabe zunächst in eine Zahl verwandelt werden muss. Mit dieser Zahl kann dann gerechnet werden und die Ergebniszahl muss wiederum in einen Buchstaben verwandelt werden.
Einige Typumwandlungen übernimmt JAVA automatisch. So lässt sich der Operator + nicht nur auf zwei Strings zu deren Verkettung zu einem gemeinsamen verwenden, sondern einer der Operanden kann auch durchaus eine Zahl sein:

```
"Die Zahl "+5+" heisst fuenf."
```

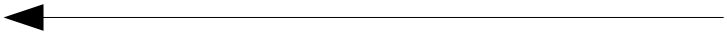
ergibt einen neuen String:

```
"Die Zahl 5 heisst fuenf."
```

Großbuchstaben

Zur Vereinfachung arbeiten wir nur mit Großbuchstaben (man könnte natürlich genau so allein mit Kleinbuchstaben arbeiten) und lassen Leerzeichen und alle Interpunktions- und Sonderzeichen weg. Dann lässt sich als Codierung der 26 Buchstaben A .. Z der ASCII-Code verwenden: A → 65, B → 66, usw. Chiffriert man mit dem Caesarbuchstaben E, dann ist mit jedem Buchstaben des Textes eine Verschiebung um (A → 0, B → 1,C → 2, D → 3,E → 4 !) vier Buchstaben innerhalb des Alphabetes durchzuführen.
Ist ein H zu chiffrieren, braucht man also dessen ASCCI-Code (72), addiert die 4 und erhält den ASCCI-Code (76) des chiffrierten Buchstaben L.
Also:

	Zeichen	ASCII	ASCII-65
Originalbuchstabe	H	72	7
Schlüsselbuchstabe	E	69	4
Chiffrebuchstabe	L	76	11



Am Ende fängt man einfach von vorn an. Wie macht man das beim Rechnen? Dafür gibt es ein einfaches Verfahren aus der Mathematik, das man von den Resten bei der ganzzahligen Division kennt: 77 modulo 10 → 7.
Also:

	Zeichen	ASCII	ASCII-65
Originalbuchstabe	Y	90	25
Schlüsselbuchstabe	E	69	4
Chiffrebuchstabe	D	68	29 mod 26 → 3

Die Umwandlungen

Für die Umwandlung in Großbuchstaben stellt die Klasse String jedem Objekt die Methode **`toUpperCase()`** zur Verfügung.

```
String text = text.toUpperCase();
```

erfüllt also das Gewünschte.

Wir brauchen dann aus dem String jeweils ein einzelnes Zeichen. Das gelingt mit der Methode **charAt(Position)** des Stringobjektes:

```
char c = text.charAt(...);
```

Der Wert vom Typ **char** dieser Methode gehört zu einem elementaren Datentyp, der sich glücklicherweise durch einen einfachen cast in die zugehörige int - Zahl verwandeln lässt:

```
int buchstabencode = (int) c;
```

Sie wird dann entsprechend der oben beschriebenen Berechnung mit dem Schlüssel zum Chiffrebuchstaben verrechnet.

Dieser muss nun allerdings wieder zurück in einen Buchstaben verwandelt werden. Auch das geht glücklicherweise mit einem einfachen cast:

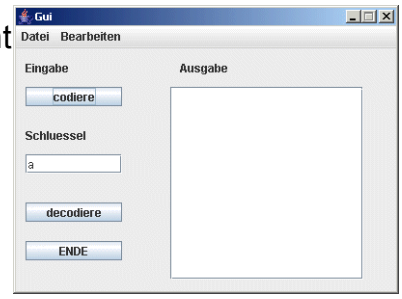
```
char chiffreBuchstabe = (char) ergebnis;
```

Wie wird wieder ein String aus den chiffrierten Buchstaben?

Dass dies überhaupt erwähnenswert ist, liegt daran, dass die Klasse String nur statische Objekte liefert; ein Stringobjekt ist also nicht veränderbar. Dafür benötigt man die Klasse StringBuffer. Instanzen von StringBuffer kennen die Methode **append(...)** zum Anhängen eines einzelnen Zeichens und liefern mit der Methode toString(), die jedes Objekt kennt, den gewünschten String.

```
StringBuffer chiffrat = new StringBuffer();
for (....) chiffrat.append(ergebnis);
return chiffrat.toString();
```

Programm zur Caesarchiffrierung mit JAVA



Eine einfache Lösung für ein JAVA – Programm, mit dem nicht nur die einfache Caesarverschlüsselung sondern auch die dabei notwendigen Dateizugriffe erledigt werden können, ist nicht einmal besonders kompliziert. Über das Layout kann man sicher streiten, das soll hier allerdings nicht Thema sein. Eine Angabe der Komponenten ist nicht schwierig, die Funktionalität naheliegend, trotzdem lohnt es sich auf einige Aspekte der Lösung einzugehen.

Datei – Dialoge

Zum Öffnen und Speichern der Dateien benötigen wir die notwendigen Dateidialoge, die uns von JAVA bereitgestellt werden. Die entsprechende Ereignismethode lautet:

```
    private void OeffnenAction(){
        JFileChooser fc = new JFileChooser("P:\\LK-Info");
        int returnVal = fc.showOpenDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            text.waehleDateiNamen((fc.getCurrentDirectory()+"\\"
                                   +fc.getSelectedFile()).getName());}
        text.dateiLesen();
        ausgabe.setText(text.gibText());
    }
```

Im Konstruktor des File – Choosers können wir als Parameter den Pfad des Netzlaufwerks übergeben, dann beginnt die Dateisuche dort. Ohne Parameter beginnt er im lokalen Ordner „Eigene Dateien“ mit der Suche. Beachten Sie die notwendige Einleitung des Sonderzeichens <*backslash*> mit dem Zeichen <*backslash*> davor. Dies ist in JAVA bei jedem Sonderzeichen innerhalb von Strings notwendig.

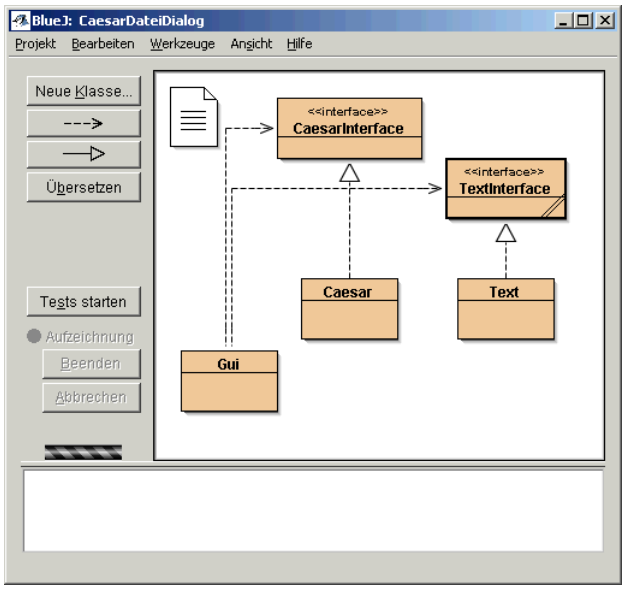
Die Klasse Text

Die Variable `text` steht für eine Instanz der Klasse Text, welche die notwendigen Operationen für die Speicherung des Textes, die Zugriffe der Gui darauf und insbesondere die Lese- und Schreib- Zugriffe auf die externen Dateien bereitstellt. Das interface lautet:

```
public interface TextInterface
{
    // gibt den gesamten Text mit \n\r dazwischen in einem String zurück.
     public String gibText();
    // setzt den gesamten Text neu.
    public void neuerText(String neu);
    // setzt den gesamten Text neu. Es muss ein Textarray übergeben werden!
    public void neuerText(String[] neu);
    // verlängert den Text um einen String.
    public void textAnhaengen(String neu);
    // setzt den Dateinamen (Ein- und Ausgabe gleich).
    public void waehleDateiNamen(String name);
    // schreibt den vorhandenen Text in die Datei. Bestehende Dateien werden
überschrieben !!!
    public void dateiSchreiben();
    // liest den Text aus der Datei. Der aktuelle Inhalt wird gelöscht.
    public void dateiLesen();
}
```

Intern arbeitet die Klasse Text mit einer ArrayList, was – wie üblich – für die Lösung aber belanglos ist.

An dieser Stelle stellt sich natürlich die Frage, weshalb der File – Chooser in Gui erzeugt wird und nicht in der Text – Klasse. Die Antwort ergibt sich aus dem Aufruf der Methode showOpenDialog, der mit `this` als Parameter das Gui – Objekt übergeben wird. Es taucht die Frage auf, ob nicht eine Modellierung besser wäre, bei der die Klasse Text eine Methode für diese Zuweisung bereitstellt und die Klasse Text dann selbst diese Aufgabe zugewiesen bekommt. In der vorliegenden Modellierung ist der Vorteil, dass die Dateibehandlung auch unabhängig von einer Gui arbeiten kann und die Gui – bezogenen Funktionen dieser zugeordnet sind.



Codieren – Aktion in Gui

In der Methode

```
private void codierenAction(){
    caesar.setzeOriginalText(ausgabe.getText());
    Character zeichen =
              Character.toUpperCase(editSchluessel.getText().charAt(0));
    if (caesar.istGrossBuchstabe(zeichen)){
        caesar.setzeSchluessel(zeichen);
        caesar.verschluessele();
        labelAusgabe.setText("codiert:");
        ausgabe.setText(caesar.zeigeText());
    } else editSchluessel.setText("Buchstaben!");
}
```

geschehen keine bemerkenswerte Dinge, daher nur kurze Hinweise darauf, wie (allein) auf das erste Zeichen des Strings im JTextField zugegriffen wird und gewährleistet wird, dass es sich dabei um einen Buchstaben handelt (Methode in Caesar). Zusätzlich wird dort auch der Buchstabe in einen Großbuchstaben konvertiert.

Absicherung in der Methode SeichernAction()

Beim Speichern könnte durch einen Bedienungsfehler ungewollt der derzeitige Inhalt in text überschrieben werden. Daher erfolgt mit dem Dialog eine Absicherung durch eine zusätzliche Abfrage:

```
private void SpeichernAction(){
    JFileChooser fc = new JFileChooser("<..Pfad ersetzen..>\\CaesarDateiDialog");
    int returnVal = fc.showSaveDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        text.waehleDateiNamen((fc.getCurrentDirectory()
             +"\\"fc.getSelectedFile()).getName()); }
    if (JOptionPane.showConfirmDialog(
        this, "Text ersetzen ?", "bestätigen" ,
           JOptionPane.OK_CANCEL_OPTION)==JOptionPane.OK_OPTION){
```

```
            text.neuerText(ausgabe.getText());
            text.dateiSchreiben();
        }
    }
```

Ebenso wird in der `beendenAction()` abgefangen, dass ungewollt beendet wird.

```
    private void beendenAction(){
        if (JOptionPane.showConfirmDialog(this, "wirklich beenden?",
           "prüfen !" , JOptionPane.OK_CANCEL_OPTION)==JOptionPane.OK_OPTION)
        System.exit(0);
    }
```

Dazu muss bei der Klasse Gui mit der JFrame - Methode

`setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);`

die normale Schließenaktion geändert werden.

Die Methode dateiLesen() in der Klasse Text

Das Lesen der Datei – ebenso wie das Schreiben in die Datei – ist von der Gui entkoppelt. Hier werden dann folgerichtig auch Fehler abgefangen.

```
    public void dateiSchreiben(){
        try {
            PrintWriter ausgabe
            = new PrintWriter(new BufferedWriter(new FileWriter(nameDatei)));
            for (int i=0; i<text[1].size();i++) {
                ausgabe.write((String)text.get(i));
                if (i<text.size()-1) ausgabe.write("\n");
            }
            ausgabe.close();
        } catch (IOException e){
            System.out.println("Fehler beim Erstellen der Datei !");
        }
    }
```

Die Datei wird nach dem vollständigen Schreiben ordnungsgemäß geschlossen. Bei einem Fehler vom Typ IOException (Eingabe – Ausgabe – Fehler) erfolgt einem Meldung in der Konsole. Das ist noch etwas unbefiedigend, da die Fehlermeldung nicht in der Gui erscheint.

Die Methode zum Lesen lautet entsprechend:

```
    public void dateiLesen(){
        try {
            BufferedReader eingabe
            = new BufferedReader(new FileReader(nameDatei));
            String zeile;
            text.clear();
            while ((zeile = eingabe.readLine()) != null){
                text.add(zeile);
            }
            eingabe.close();
        } catch (IOException e){
            System.out.println("Fehler beim Lesen der Datei !");
        }
    }
```

1 text ist Instanzvariable in der Klasse Text: private ArrayList<String> text; – also eine typisierte ArrayList.

Analyse und Ausgabe der Buchstabenhäufigkeit

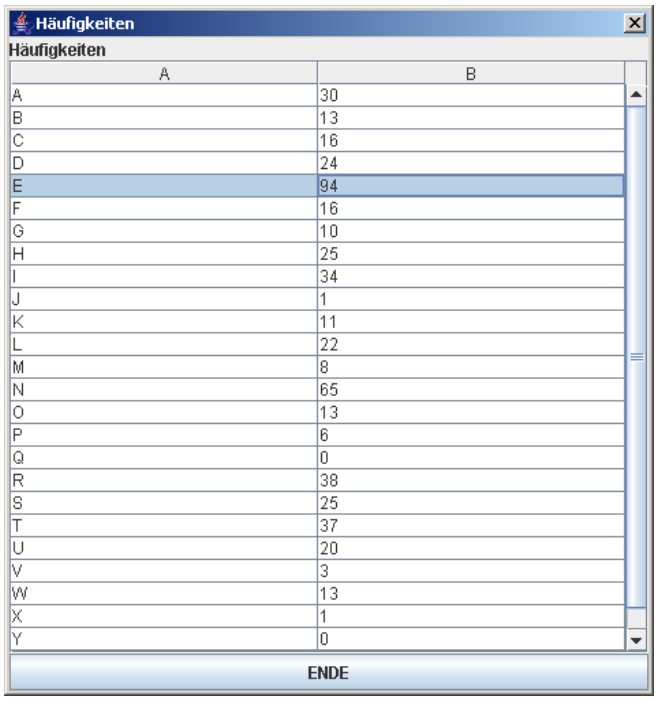
Die Analyse der Buchstabenhäufigkeit, sollte auch aus der Oberfläche heraus angestoßen werden können. Dazu ist einmal über die Benutzerschnittstelle nachzudenken, andererseits aber auch darüber, wo die Berechnung anzusiedeln ist.

Der Benutzer wird dafür einen Menüpunkt im Menü erwarten und zwar im „Bearbeiten“ – Menü, dem wir ein Item „Häufigkeiten“ hinzufügen.

Die Darstellung könnte zwar im vorhandenen Fenster erfolgen, also z.B. an Stelle der JTextArea, das Fenster könnte auch in der Größe erweitert werden, so dass auch die Häufigkeiten dargestellt werden können.

Die bessere Variante ist aber ein zusätzliches Fenster (→ Kohärenz), das nur diese eine Aufgabe hat und wieder geschlossen werden kann, wenn die Häufigkeiten nicht mehr interessant sind.

Dazu verwenden wir ein Dialogfenster mit dem Titel AusgabeHaeufigkeiten, das von JDialog erbt. Im Beispiel zeigt es die Häufigkeiten eines deutschen Textes ohne Verschiebung an und der häufigste Buchstabe, das e, ist hervorgehoben. Klickt man auf den Button Ende, wird das Fenster geschlossen. Die Elemente sind mit BorderLayout positioniert.




```
    public AusgabeHaeufigkeiten(JFrame owner, String title, boolean modal,
                        int[] haeufigkeiten) {
        super(owner, title, modal);
        this.haeufigkeiten=haeufigkeiten;
        setLayout(new BorderLayout());
        labelHaeufigkeiten = new JLabel("Häufigkeiten");
        add(labelHaeufigkeiten, BorderLayout.NORTH);
        endeButton = new JButton ("ENDE");
        add(endeButton, BorderLayout.SOUTH);
```

Der Dialog bekommt beim Erzeugen das Häufigkeiten – Array übergeben und verwendet intern das AbstractTableModel, das wir schon mehrfach benutzt haben.

Die Berechnung stellt die Klasse Caesar mit der folgenden Methode bereit:

```
    // erstellt eine Häufigkeitsanalyse der Buchstaben des Codes.
    // gibt die Ergebnisse in einem Array zurück.
    public int[] haeufigkeiten(){
        int[] haeufigkeiten= new int[26];
        Character c;
        if (code!=null){
            for (int i=0; i<code.length();i++){
                c=code.charAt(i);
                if (istGrossBuchstabe(c))
haeufigkeiten[Character.getNumericValue(c)-10]++;
            }
        }
        return haeufigkeiten;
    }
```