

4 Codierung nach Viginere (Lösung)

4.1 Einführung

Blaise de Vigenère lebte von 1523 bis 1596 in Frankreich und war nach dem Studium bei verschiedenen Herren im diplomatischen Dienst eingestellt. Bei einer diplomatischen Mission in Rom entdeckte er in einem Archiv die Arbeiten von Alberti und anderer Kryptologen. Schnell wurde aus dem anfangs nur praktischen Interesse ein Lebensziel: diese Schriften alle zu studieren und ein neues, mächtigeres Chiffriersystem zu entwickeln. 1570 gab er den Dienst auf und widmete sich seinen Interessen. 1580 veröffentlichte es sein Werk *Traictè de Chiffres*. Das Buch gibt einen genauen Stand der Kryptographie seiner Zeit wieder und enthält außerdem Goldmacherrezepte, Alchemie und japanische Ideogramme.



Abbildung 4.1: Blaise de Viginere

Als Grundlage der Viginere-Verschlüsselung dient die so genannte Viginere-tabelle. In der ersten Zeile steht das Klartextalphabet, darunter die Geheimtextalphabete jeweils um einen Buchstaben versetzt. Mit Hilfe dieser Tabelle konnte man im 15. Jahrhundert Botschaften folgendermaßen verschlüsseln.

Klar	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
1	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
2	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	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I

10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	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
26	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

Abbildung 4.2: Viginere-Quadrat mit hervorgehobenen Zeilen, die durch das Schlüsselwort LICHT bestimmt sind.

Die Abbildung 4.3 zeigt die Verschlüsselung des Klartexts „truppenabzugnachosten“

Schlüsselwort	L	I	C	H	T	L	I	C	H	T	L	I	C	H	T	L	I	C	H	T	L
Klartext	t	r	u	p	p	e	n	a	b	z	u	g	n	a	c	h	o	s	t	e	n
Geheimtext	E	Z	W	W	I	P	V	C	I	S	F	O	P	H	V	S	W	U	A	X	Y

Abbildung 4.3: Verschlüsselung mit Hilfe des Schlüsselworts LICHT

Man benötigt ein Schlüsselwort, hier z.B. LICHT und schreibt dieses über die zu verschlüsselnde Botschaft, hier Klartext genannt. Um also den ersten Buchstaben "t" der Botschaft "truppenabzugnachosten" zu codieren, findet man in der Zeile "L" (von LICHT) und in der Spalte "t" (von truppen...) den entsprechenden Geheimbuchstaben "E". Um den 2. Buchstaben "r" zu verschlüsseln, findet man in der Zeile "I" und in der Spalte "r" den Geheimbuchstaben "Z". Und so geht es weiter. Jeder kann die Viginere-Tabelle besitzen, aber um den Geheimtext zu entschlüsseln, braucht man auf jeden Fall das Schlüsselwort "Licht". Über 300 Jahre, bis 1853, dauerte es, bis Babbage ein Verfahren fand, diese Verschlüsselung zu knacken ohne Kenntnis des Schlüsselworts mit Hilfe einer Häufigkeitsanalyse von Buchstaben.

4.2 Definieren der Schnittstelle

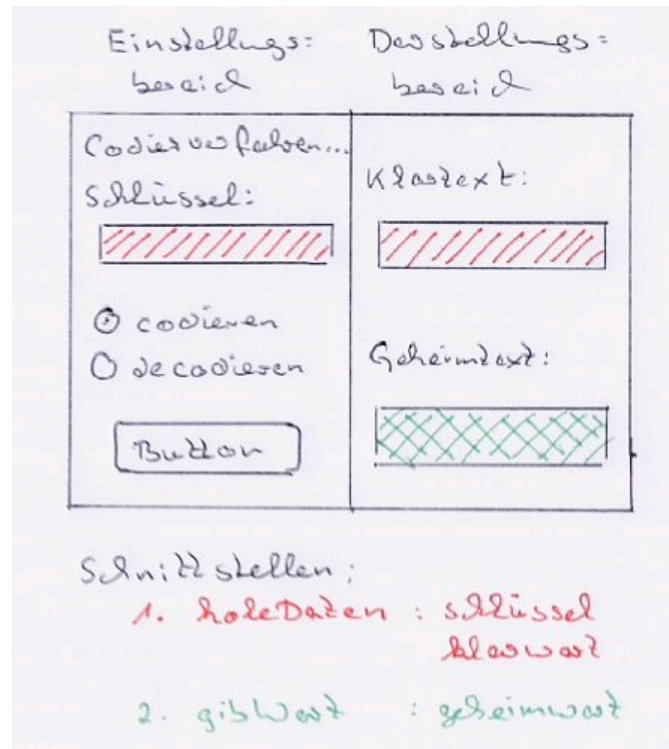


Abbildung 4.4: Definieren der Schnittstelle

Die Abbildung 4.4 zeigt, wie unsere grafische Benutzeroberfläche gestaltet sein könnte.

Unabhängig vom Aussehen der GUI benötigen wir zur Codierung eines Textes das Schlüsselwort und das Klarwort. Diese beiden Worte sollen vom Benutzer in entsprechenden Textfelder eingegeben werden. Die Klasse *Viginere* ist also verantwortlich, dass sie sich diese Daten aus der GUI selber holt. Dafür wird eine Methode *holeDaten()* implementiert, die diese Aufgabe zu erledigen hat.

Die Klasse *Viginere* codiert nun mit Hilfe des Schlüsselworts das eingegebene Klarwort. Das entsprechende Geheimwort gibt die Klasse *Viginere* mit Hilfe der Methode *gibWort()* wieder zurück.

Somit hat die Klasse *Ansicht* das Geheimwort und kann dann es in das dafür vorgesehene Textfeld einlesen.

Die Klasse *Viginere* hat also die folgenden öffentlichen Methoden: *holeDaten()* und *gibWort()*. Nur diese beiden Methoden können von der Klasse *Ansicht* gesehen und somit auch verwendet werden.

4.3 Die Klasse *Viginere*

4.3.1 Verschlüsselung

Wie in Abbildung 4.3 angedeutet, benötigt die Klasse *Viginere* das Klarwort und das Schlüsselwort und ermittelt daraus das Geheimwort. Allerdings können wir Klarwort und Geheimwort in einem einzigen Datenfeld speichern. Somit besitzt die Klasse *Viginere* die beiden Datenfelder *wort* und *schluessel*.

Viginere
wort schluessel
Methoden...

Abbildung 4.5: Datenfelder der Klasse *Viginere*

Diese beiden Datenfelder müssen nun mit entsprechenden Werten belegt werden. Diese Werte holt sich die Klasse *Viginere* mit Hilfe der Methode *holeDaten()* aus den Textfeldern der GUI. In der Methode *codiereWort()* wird aus dem Klartext der Geheimtext berechnet. Mit Hilfe der Methode *gibWort()* gibt die Klasse *Viginere* nun den Geheimtext wieder zurück an die Klasse *Ansicht*, die ihn in das entsprechende Textfeld einsetzt.

Viginere
wort schluessel
holeDaten() codiereWort() holeVerschiebung() gibWort()

Abbildung 4.6: Datenfelder und Methoden der Klasse *Viginere*

In der Methode *codiereWort()* findet nun die Verschiebung des Klartextalphabets statt. Die Anzahl der Stellen, um die das Wort verschoben wird, hängt vom jeweiligen Schlüsselwort ab. Diese Anzahl der Stellen wird mit Hilfe der Methode *holeVerschiebung()* ermittelt und in eine lokale Variable *versch* gelesen. Der Rest der Methode *codiereWort()* ist analog der Klasse *Caesar* implementiert, muss also hier nicht besonders erklärt werden.

```

private void codiereWort()
{
    StringBuffer wortBuffer;
    int wortLaenge, x, versch;
    char ch;

    wortBuffer = new StringBuffer();
    wort = wort.toUpperCase();
    wortLaenge = wort.length();
    for (int i = 0; i < wortLaenge; i++) {
        ch = wort.charAt(i);
        x = (int) ch;
        x = x - 65;
        versch = holeVerschiebung(i);
        x = x + versch;
        x = x % 26;
        x = x + 65;
        wortBuffer.append((char)(x));
    }
    wort = wortBuffer.toString();
}

```

Abbildung 4.7: Die Methode *codiereWort()*

Wie im Eingangsbeispiel erwähnt, wird über dem Klartext das Schlüsselwort so oft geschrieben, bis jeder Klartextbuchstabe unter einem Schlüsselwortbuchstaben steht. Im Datenfeld *schluessel* steht bis jetzt nur das Wort „Licht“, es sollte aber hier eine entsprechende Wortwiederholung des Worts „Licht“ stehen. Diese Wortwiederholung wird in der Methode *holeVerschiebung()* in der *for*-Schleife erzeugt:

```

wortSchluessel = new StringBuffer();
schluessel = schluessel.toUpperCase();
schluesselLaenge = schluessel.length();
wortLaenge = wort.length();
for (int i = 0; i < wortLaenge; i++) {
    ch = schluessel.charAt(i % schluesselLaenge);
    y = (int) ch - 65;
    wortSchluessel.append((char) (y + 65));
}
schluessel = wortSchluessel.toString();

```

Abbildung 4.8: Wiederholung des Schlüsselworts innerhalb der Methode *holeVerschiebung()*

Bemerkung:

Die Erzeugung des Strings „LICHTLICHTLIC...“ bei **jedem** Aufruf der Methode *holVerschiebung()* ist überflüssig. Es ist besser, diese Anweisungen in

einer separaten Methode einmal auszuführen, so dass der String *schluessel* immer zur Verfügung steht.

Das Datenfeld *schluessel* der Klasse *Viginere* enthält also nun einen String, der aus genauso vielen Zeichen „LICHTLICHTLIC...“ besteht wie der Klartext.

Beim Aufruf der Methode *holeVerschiebung()* wird als Parameter die Stelle desjenigen Buchstabens übergeben, der gerade verschlüsselt werden soll. Mit Hilfe dieses Parameterwertes findet man den entsprechenden Buchstaben in dem Schlüsselwort *schluessel*. Dessen ASCII-Code wird um 65 verringert und in die Variable *versch* gelesen.

```
ch = schluessel.charAt(n);
versch = (int) ch - 65;
return versch;
```

Abbildung 4.9: Berechnung der *versch* innerhalb der Methode *holeVerschiebung()*

Diese beiden Quelltextfragmente müssen nun zur Methode *holeVerschiebung()* zusammengesetzt werden:

```
private int holeVerschiebung(int n)
{
    StringBuffer wortSchluessel;
    int wortLaenge, schluesselLaenge, y, versch;
    char ch;

    ..... obige Anweisungen .....
}
```

Abbildung 4.10: Die Methode *holeVerschiebung()*

Dieser Wert wird wieder der Methode *codiereWort()* zurückgegeben, so dass nun die entsprechende Verschiebung durchgeführt werden kann.

Ich will am Eingangsbeispiel durchspielen, was in den beiden Methoden *codiereWort()* und *holeVerschiebung()* der Klasse *Viginere* berechnet wird:

i	1	2	3	4	5	6	7	8	9
ch	t	r	u	p	p	e	n	a	b
x	19	17	20	15	15	4	13	0	1
versch holV(i)	11	8	2	7	19	11	8	2	7

$x = x + \text{versch}$	30	25	22	22	34	15	21	2	8
$x = x \% 26$	4	25	22	22	8	15	21	2	8
$x = x + 26$	69	90	87	87	73	80	86	67	73
	E	Z	W	W	I	P	V	C	I

Abbildung 4.11: Verschlüsselung in der Klasse *Viginere*

Übung:

Das Projekt *Viginere01a* behandelt die Verschlüsselung nach Viginere.

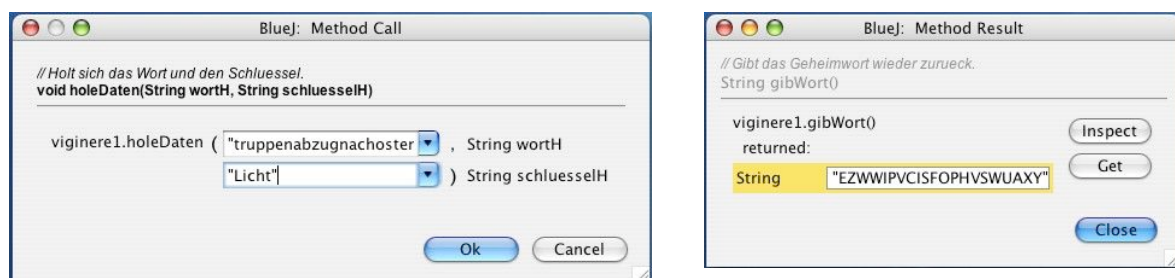


Abbildung 4.12: Die Methoden *holeDaten()* und *gibWort()*

4.3.2 Entschlüsselung

Am leichtesten lässt sich die Entschlüsselung mit Hilfe der folgenden Tabelle durchspielen, was genau in den Methoden *decodiereWort()* und *holeVerschiebung()* der Klasse *Viginere* durchgeführt wird:

i	1	2	3	4	5	6	7	8	9
ch	E	Z	W	W	I	P	V	C	I
x	4	25	22	22	8	15	21	2	8
versch holV(i)	11	8	2	7	19	11	8	2	7
versch = - versch	-11	-8	-2	-7	-19	-11	-8	-2	-7
$x = x + \text{versch}$	-7	17	20	15	-11	4	13	0	1
$x = x \% 26$	-7	17	20	15	-11	4	13	0	1

if (x < 0) x = x + 26	19	17	20	15	15	4	13	0	1
x=x+65	84	82	85	80	80	69	78	65	66
	t	r	u	p	p	e	n	a	b

Abbildung 4.13: Entschlüsselung in der Klasse *Viginere*

Ein Vergleich mit der Tabelle in Abbildung 4.10 zeigt, dass im Algorithmus der Entschlüsselung nur die beiden Anweisungen

`versch = - versch` und `if (x < 0) {x = x + 26}` hinzugefügt werden.

Übung:

Das Projekt *Viginere01b* behandelt die Entschlüsselung nach Viginere.

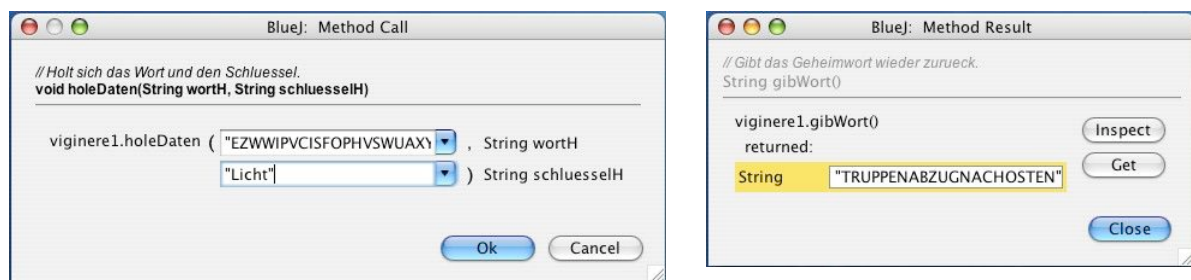


Abbildung 4.14: Die Methoden *holeDaten()* und *gibWort()*

4.3.3 Zusammenfassung

Nun sollten die Verschlüsselung und die Entschlüsselung zu einer einzigen Klasse zusammengefasst werden. Hierzu führe ich eine Variable *codier* vom Typ *boolean* ein. Falls diese den Wert *false* hat, werden die zusätzlichen Anweisungen bei der Entschlüsselung ausgeführt.

Viginere
wort schluessel codier
holeDaten()

```
codiereWort()  
gibWort()  
holeVerschiebung()
```

Abbildung 4.15: Datenfelder und Methoden der Klasse *Viginere*

Übung:

Das Projekt *Viginere01c* behandelt die Verschlüsselung/Entschlüsselung nach Viginere.

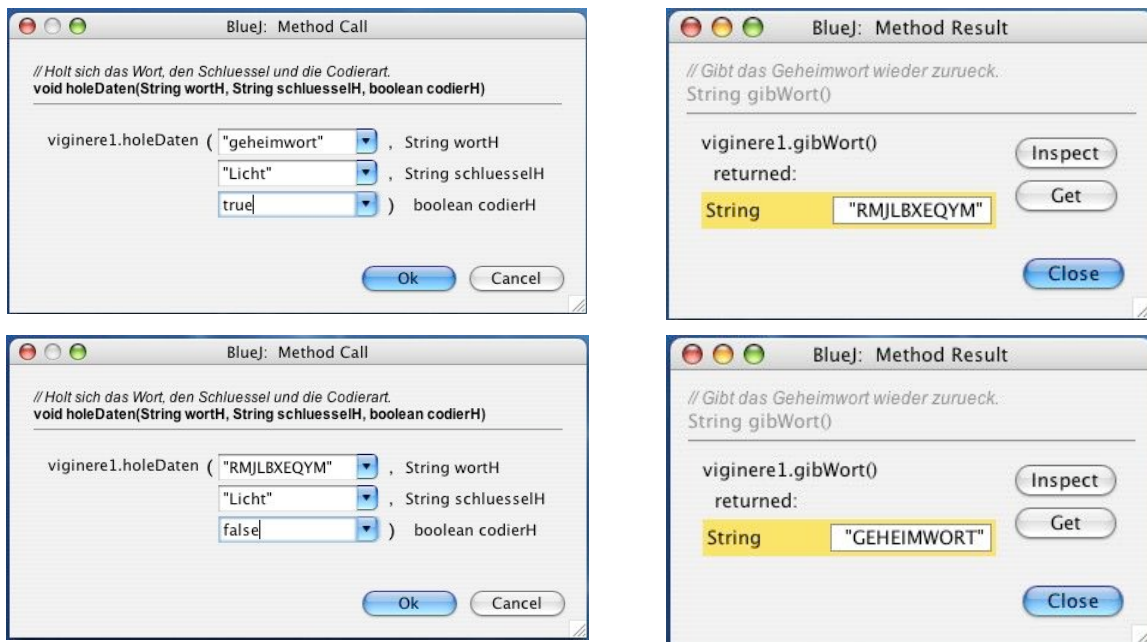


Abbildung 4.16: Die Methoden *holeDaten()* und *gibWort()* beim Verschlüsseln bzw. Entschlüsseln

4.4 Die grafische Oberfläche

4.4.1 Gestaltung

Die grafische Benutzeroberfläche übernehme ich aus Kapitel 1 Abbildung 1.16.



Abbildung 4.17: Beispiel für die GUI von *Viginere02a*

Es wurden folgende Veränderungen durchgeführt:

1. Name des Fensters
2. Name des Labels *schluesselL* und Text
3. Name des Textfeldes *schluesselTF*

Übung:

Das Projekt *Viginere02a* behandelt die Oberfläche.

4.4.2 Reaktionen

Nun müssen wir noch die Reaktionen programmieren, wenn auf die Radiobutton bzw. auf den Button geklickt wird.

Die Reaktionen auf die beiden Radiobuttons sind recht einfach. Wird codiert, ist das obere Textfeld im Darstellungsbereich für den Klartext, das untere für den Geheimtext vorgesehen. Beim Decodieren wird entsprechend umgekehrt im oberen Textfeld der Geheimtext eingeben und im unteren der Klartext wieder angezeigt. Dementsprechend müssen die Labels die richtige Beschriftung wider geben.

```
private void codierRBIItemStateChanged(ItemEvent evt)
{
    if (codierRB.isSelected()) {
        eingabeL.setText("Klartext:");
        ausgabeL.setText("Geheimtext:");
    }
}
```

```
private void decodierRBIItemStateChanged(ItemEvent evt)
```

```
{
    if (decodierRB.isSelected()) {
        eingabeL.setText("Geheimtext:");
        ausgabeL.setText("Klartext:");
    }
}
```

Abbildung 4.18: Methoden zur Beschriftung der beiden Textfelder

Die Reaktion auf den Button lässt sich weit gehend aus Projekt *Caesar03d* übernehmen. Sie unterscheidet sich nur in der Verwendung der zusätzlichen Variablen *codieren*. Ihr Wert ist *true* bzw. *false* je nach Wahl des Radiobuttons.

```
private void uebersetzBActionPerformed(ActionEvent evt)
{
    String wort, schluessel;
    boolean codieren = true; //gerade aktivierter Radiobutton

    Viginere einViginere = new Viginere();
    wort = eingabeTF.getText();
    schluessel = schluesselTF.getText();
    if (codierRB.isSelected()) codieren = true;
    if (decodierRB.isSelected()) codieren = false;
    einViginere.holeDaten(wort, schluessel, codieren);
    wort = einViginere.gibWort();
    ausgabeTF.setText(wort);
}
```

Abbildung 4.19: Die Methode *uebersetzBActionPerformed()* in der Klasse *Ansicht*

Übung:

Das Projekt *Viginere02b* behandelt die Oberfläche mit den Reaktionen.

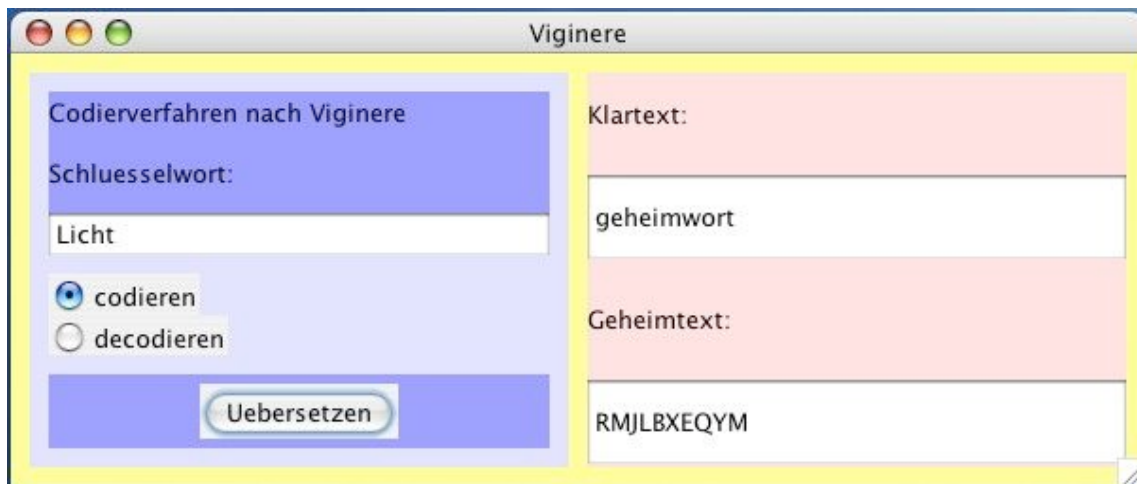


Abbildung 4.20: Das Projekt *Viginere02b* beim Verschlüsseln



Abbildung 4.21: Das Projekt *Viginere02b* beim Entschlüsseln

4.5 Eine stand-alone Applikation

Bis jetzt läuft unser Programm nur in der Entwicklungsumgebung **BlueJ**. Wir wollen es jedoch als selbstständiges Programm an dritte Personen weitergeben können. Hierzu benötigen wir eine so genannte *main()*-Methode. Wird diese Methode beim Aufruf des Programms gefunden, so läuft der Rest sozusagen von allein ab.

Ich lege diese Methode *main()* aus didaktischen Gründen in eine eigene Klasse *Codierung*.

```
public class Codierung
{
    Ansicht ansicht;

    public Codierung()
```

```
{
    ansicht = new Ansicht();
}

public static void main(String[] args)
{
    Codierung q = new Codierung();
}
}
```

Abbildung 4.22: Quellcode der Klasse *Codierung*

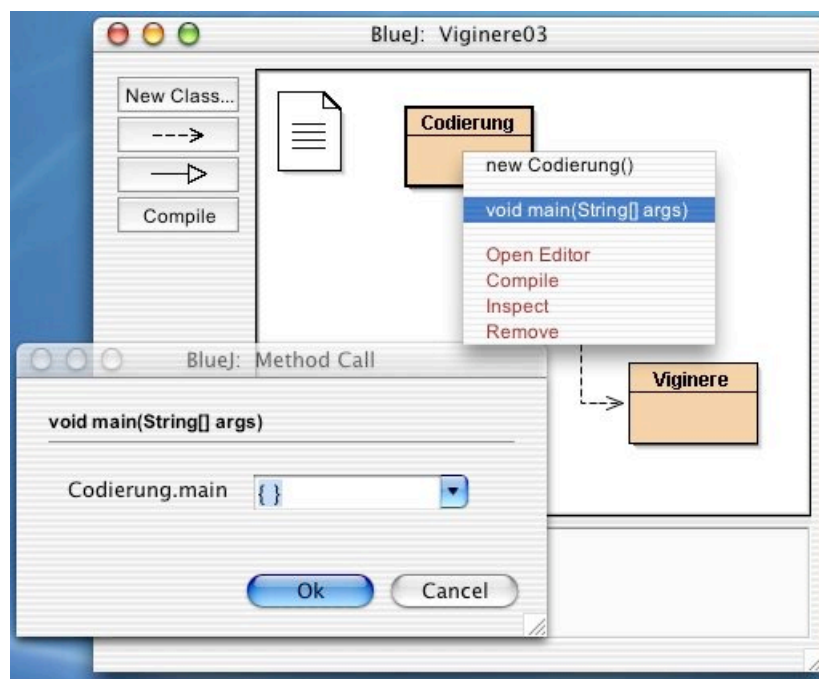


Abbildung 4.23: Klassendiagramm von *Viginere03* mit dem Aufruf der Methode *main()*



Abbildung 4.24: Das Programm *Viginere03* beim Verschlüsseln



Abbildung 4.25: Das Programm *Viginere03* beim Entschlüsseln

Übung:

Aus dem Projekt *Viginere03* kann eine stand-alone-Applikation erstellt werden.