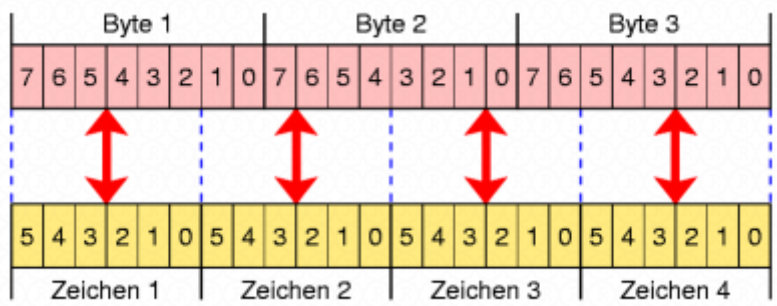
Base64-Code

Base64 ist ein Verfahren zur Kodierung von beliebigen 8-Bit-Binärdateien (also Daten, die aus Bytes bestehen) in eine Zeichenfolge, die nur aus „druckbaren“ Zeichen besteht. Diese druckbaren Zeichen sind die Groß- und Kleinbuchstaben des Alphabets sowie die zehn Ziffern und zusätzlich noch die beiden Zeichen + und /.

Das *Base64*-Verfahren findet im Internet Anwendung und wird dort zum Beispiel zum Versenden von E-Mail-Anhängen (Bild- oder Musikdateien) verwendet. Der Grund dafür liegt darin, dass in E-Mail-Programmen nicht beliebige Bytes verschickt werden können, weil einige bestimmte Bytes eine besondere Bedeutung für den Datentransport haben. Allerdings werden die normalen Buchstaben und Ziffern grundsätzlich problemlos übertragen. Durch die Kodierung steigt der Platzbedarf des Datenstroms um (ziemlich genau) 33%.



Zur Kodierung werden jeweils drei Byte des Bytestroms (= 24 Bit) in vier 6-Bit-Blöcke aufgeteilt. Jeder dieser 6-Bit-Blöcke bildet eine Zahl von 0 bis 63. Der Name des Algorithmus erklärt sich durch ebendiesen Umstand: Jedem Zeichen des kodierten Datenstroms lässt sich eine Zahl von 0 bis 63 zuordnen (siehe Tabelle). Diese Zahlen werden anhand der nachfolgenden Umsetzungstabelle in „druckbare *ASCII*-Zeichen“ umgewandelt und wieder (als normale 8-bit-Zeichen) ausgegeben.

Falls die Gesamtanzahl der Eingabebytes nicht durch drei teilbar ist, wird die zu kodierende Eingabe am Ende mit aus Nullbits bestehenden Füllbytes aufgefüllt, sodass sich eine durch drei teilbare Anzahl an Bytes ergibt.

Hinweis: Auch bei bloßen Texteingaben ist die Anzahl der Eingabezeichen (mit Leer-, Satz- sowie Zeilensprungzeichen) nicht unbedingt identisch mit der Anzahl der Eingabebytes, weil z.B. deutsche Umlaute im utf-8-Zeichensatz durch zwei Bytes dargestellt werden.

Um dem Dekodierer mitzuteilen, wie viele Füllbytes angefügt wurden, werden diejenigen 6-Bit-Blöcke, die vollständig aus Füllbytes entstanden sind, mit dem

Zeichen = kodiert (also nicht durch den Buchstaben A, welcher normalerweise dem Sextett 000000 entspricht – siehe nachfolgende Codetabelle!). Somit können am Ende einer Base64-kodierten Datei null, ein oder zwei =-Zeichen auftreten. Anders gesagt, es werden so viele =-Zeichen angehängt, wie Füllbytes angefügt worden sind.

Bei einer zu kodierenden Eingabe mit Byte beträgt der Platzbedarf für den *Base64*-kodierten Inhalt $z = 4 \cdot \left\lceil \frac{n}{3} \right\rceil$ Zeichen. Die Klammern um den Bruch stehen für die aufrundende Ganzzahldivision.

Einfaches Beispiel:
Der Originaltext: „Info ist toll“ soll *Base64*-kodiert werden.

Die zugehörigen 13 Codezahlen der Zeichen sind dezimal:
73 110 102 111 32 105 115 116 32 116 111 108 108

Als achtstellige Binärzahlen hintereinander gereiht ergeben sich 13 Binärzahlen. Zusätzlich hängt man noch zwei Nullbytes an (weil 13 nicht durch 3 teilbar ist), so dass sich 15 Bytes ergeben:
0100 1001 | 0110 1110 | 0110 0110 | 0110 1111 | 0010 0000 | 0110 1001 |

0111 0011 | 0111 0100 | 0010 0000 | 0111 0100 | 0110 1111 | 0110 1100 |

0110 1100 | 0000 0000 | 0000 0000

Interpretiert man diese Folge von Nullen und Einsen als sechsstellige Binärzahlen, so erhält man folgende 20 (sechsstellige) Binärzahlen:
010010 | 010110 | 111001 | 100110 | 011011 | 110010 | 000001 | 101001 |

011100 | 110111 | 010000 | 100000 | 011101 | 000110 | 111101 | 101100 |

011011 | 000000 | 000000 | 000000
Dabei sind nur die beiden letzten Sextette vollständig aus Nullbytes entstanden. Diese beiden müssen also im Base64-Code durch zwei =-Zeichen kodiert werden.

Im Dezimalsystem entspricht das folgenden 20 Zahlen:
18 22 57 38 27 50 1 41 28 55 16 32 29 6 61 44 27 0 0 0

Laut der nachfolgenden Tabelle für den *Base64*-Zeichensatz entspricht das den folgenden 20 „druckbaren“ Zeichen:
SW5mbyBpc3QgdG9sbA==
(beachte, dass die letzten drei Nullen unterschiedlich kodiert werden!)

Base64-Zeichensatz

dez	binär		**dez.**	**binär**		**dez.**	**binär**		**dez.**	**binär**	
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/

Weiteres Beispiel:
Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und Quark

Der (mit Leerzeichen) 64 Zeichen lange Text ist *UTF-8*-kodiert 68 Byte lang, weil das ß und die Umlaute jeweils eine Länge von zwei Byte haben, und wird mit der Umwandlung in den *Base64*-Code zur folgenden 92 Zeichen langen *Base64*-Zeichenkette:

UG9seWZvbiB6d2l0c2NoZXJuZCBhw59lbiBNw6R4Y2hlbnMgVsO2Z2VsIFLDvGJlbiwgSm9naHVydCB1bmQgUXVhcms=

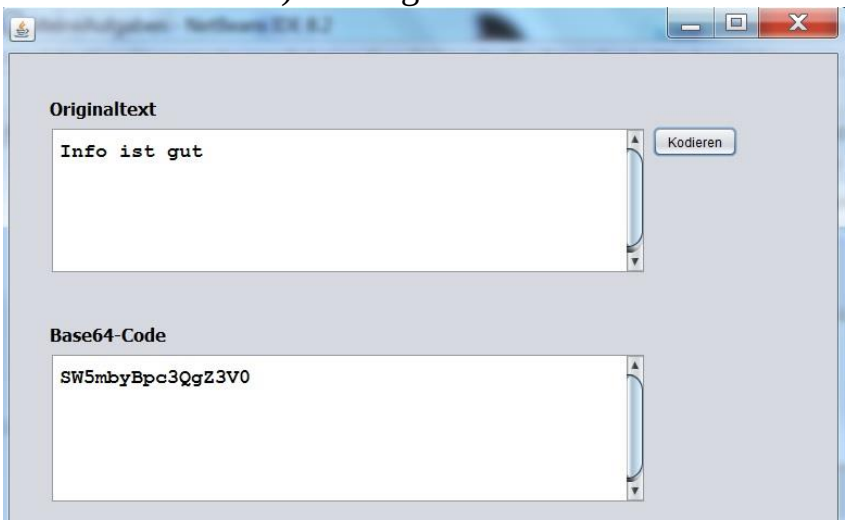
Aufgabe 1
Die Javamethode **`Integer.toBinaryString(int n)`** liefert (als String) die Binärdarstellung der Zahl n; allerdings nur mit minimal notwendiger Stellenzahl, also nicht unbedingt achtstellig.

Schreibe deshalb eine Methode **`String dualZahlwort(int n)`**, welche für den Parameter n mit $0 \leq n \leq 255$ eine genau achtstellige Dualzahl als String liefert! Oben erwähnte vordefinierte Javamethode sollte dabei genutzt werden.

Aufgabe 2
Schreibe eine Methode **`int binaer6ToDez(String dualzahl)`**, welche für eine 6-stellige Dualzahl (übergeben als Stringparameter) die zugehörige Dezimalzahl (vom Typ *int*) liefert!

Aufgabe 3
Schreibe eine Methode **`char base64Zeichen(int n)`**, welche für den Parameter n mit $0 \leq n \leq 63$ das zugehörige *base64*-Zeichen liefert!

Aufgabe 4
Schreibe ein Programm, welches zwei Textareas namens *taOriginal* und *taBase64* enthält. Der Benutzer kann einen beliebigen Text in *taOriginal* hineinschreiben und auf Button-Klick wird dieser entsprechend kodiert in der Textarea *taBase64* ausgegeben.
Tip: Setze zunächst voraus, dass die Anzahl der Eingabe-Bytes (einschließlich Leerzeichen) im Originaltext durch 3 teilbar ist!



Aufgabe 5
Wie Aufgabe 4, nur soll diesmal nicht kodiert sondern dekodiert werden. Benutze dafür einen zweiten Button!

Lösung Aufgabe 1

```
private String dualZahlwort(int n)  {
   String zahlwort = Integer.toBinaryString(n);
   int laenge = zahlwort.length();
   int differenz = 8 - laenge;
   for (int i=1; i<=differenz; i++) zahlwort = '0' + zahlwort;
   return zahlwort;
}
```

Lösung Aufgabe 2

```
private int binaer6ToDez(String dualzahl)  {
   int dezsum=0;
   if (dualzahl.charAt(0) == '1') dezsum = dezsum + 32;
   if (dualzahl.charAt(1) == '1') dezsum = dezsum + 16;
   if (dualzahl.charAt(2) == '1') dezsum = dezsum + 8;
   if (dualzahl.charAt(3) == '1') dezsum = dezsum + 4;
   if (dualzahl.charAt(4) == '1') dezsum = dezsum + 2;
   if (dualzahl.charAt(5) == '1') dezsum = dezsum + 1;
   return dezsum;
}
```

Lösung Aufgabe 3

```
private char base64Zeichen(int n){
   char ch;
   if (n <= 25) ch = (char) (n+65);
   else if (n <= 51) ch = (char) (n+71);
        else if (n <= 61) ch = (char) (n-4);
             else if (n == 62) ch = '+';
                  else ch = '/';
   return ch;
}
```

Lösung Aufgabe 4 *// mit einer durch 3 teilbaren Anzahl von Eingabe-Bytes*

```
private void jButton1MouseClicked(java….)  {
   String originalText = taOriginal.getText();
   int laenge = originalText.length(); // Vorsicht: Anzahl Bytes?
   String binaertext = "";
   for (int i = 0; i < laenge; i++)
      binaertext = binaertext + dualZahlwort((int) originalText.charAt(i));
   String base64Text = "";
   String sechsZiffern;
   for (int i=0; i < binaertext.length(); i=i+6)   {
        sechsZiffern = binaertext.substring(i, i+6);

        base64Text = base64Text + base64Zeichen(binaer6ToDez(sechsZiffern));
   }
   taBase64.setText(base64Text);
}
```