Hochschule Harz	FB Automatisierung und Informatik
Programmierung 2	Dipl.-Inf. Dipl.-Ing. (FH) M. Wilhelm
Aufgabe 03:	„Programmierung 2“ für MI / WI Thema: **Swing, GridBagLayout und Bit-Funktionen**

Versuchsziele

Kenntnisse in der Anwendung von:

- Erstellen einer Swing-Anwendung
- BitProgrammieren von Bit-Operationen
    - Bit setzen
    - Bit löschen
    - Bits abfragen

Aufgabe 03

Swing:

Die Swing-Oberfläche wird vorgegeben:

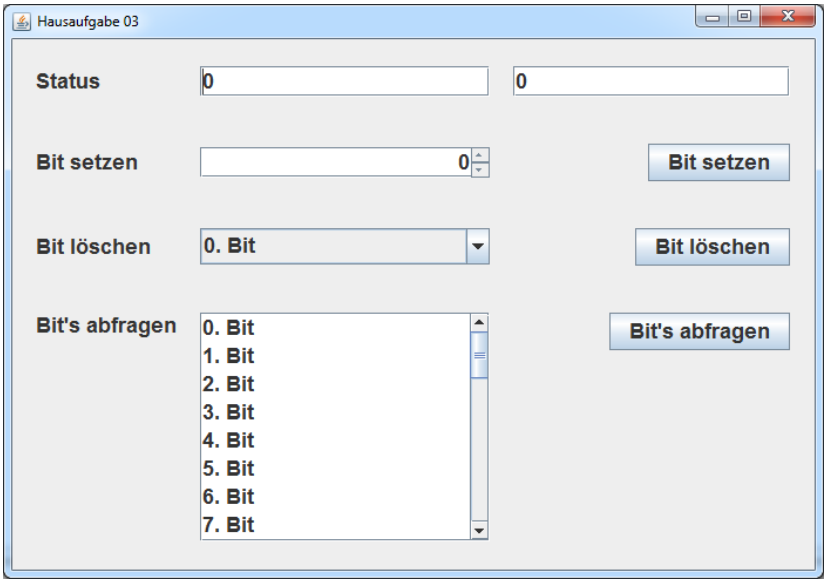

Abbildung 1 Musterlösung

Das Programm hat eine Instanz der Klasse „BitValue“ als globale Variable. Alternativ kann man auch das Singleton-Pattern anwenden.
Schalter-Aktionen:

- Bit setzen
    - Das Bit des JSpinners soll in der Instanz bitValue gesetzt werden.
- Bit löschen
    - Das Bit der ComboBoxs soll in der Instanz bitValue gelöscht werden.
    - Bitte den Anhang buw. Das Beispielsprogramm analysieren
- Bits abfragen
    - Die markierten Bits der JList sollen in der Instanz bitValue abgefragt werden.
    - Das Ergebnis soll mittels MessageBox ausgegeben werden. Jeweils für den then und else-Fall.
- Auf meiner Homepage steht eine Musterlösung als Exe-Datei zur Verfügung. Diese ist nur lauffähig unter Windows. Gegebenenfalls Virtual-Box installieren.

1. Teilaufaufgaben: Projekt erstellen

- Erstellen Sie ein neues Eclipse–Projekt:
  - Projektname: Aufgabe03
  - Klassenname: Aufgabe03
- Erstellen Sie eine neue Klasse
  - Menü File, Eintrag New, Eintrag class
  - Name: Aufgabe03
- Kopieren des Rahmensprogramm in die Datei
  - Quelle: http://mwilhelm.hs-harz.de/scripte/prog2/Aufgabe03.java

2. Teilaufaufgaben: Klasse erstellen

Die Klasse BitValue kapselt ein Zahlenwert, dessen Wert durch zwei Funktionen verändert werden kann. Die Funktion checkBits testet, ob Bits gesetzt sind. Zwei weitere Methoden dienen der Ausgabe als String bzw. als Binärtext. Das Programm soll ein int-Wert verwalten. Da aber Java nur vorzeichenbehaftete Datentypen hat, muss ein long verwendet werden. Nach außen, also JSpinner, ComboBox und JList, wird ein int „vorgegaukelt“. Das heißt, dass die Bits null bis einunddreißig zur Auswahl stehen.

- Erstellen Sie die Klasse „**BitValue**“
  - Variable
    - Name: bitValue
    - Datentyp:long
  - Konstruktor
    - setzen der Variable
  - Methoden
    - setBit
      - Parameter:
        - Name: bitValues
        - Typ: long
      - Setzt die Bits des Parameterwertes.
      - Rückgabe: kein
    - delBit
      - Parameter:
        - Name: bitValues
        - Typ: long
      - Löscht die Bits des Parameterwertes
      - Rückgabe: kein
      -
    - checkBits
      - Parameter:
        - Name: bitValues
        - Typ: long
      - Prüft, ob die gesamten Bits des Parameterwertes gesetzt sind.
      - Rückgabe: true oder false
    - toBinString
      - Parameter: keine
      - Rückgabe: Binärdarstellung der Variable bitValue
        - Benutzen Sie irgendeine Methode der Klasse L…
    - toString
      - Parameter: keine
      - Rückgabe: Binärdarstellung der Variable bitValue
      - Benutzen Sie irgendeine Methode der Klasse L…

Rahmen auch im Internet erhältlich:

```
//Titel:        3. Hausaufgabe, Programmierung 2
//Version:      1,0
//Copyright:    Copyright (c) 2013
//Autor:        M. Wilhelm
//Organisation: HS-Harz
//Beschreibung: GridBagLayout und Bit-Operationen

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class Aufgabe03 extends JFrame {

   public static final long serialVersionUID=1;


  JLabel  lStatus = new JLabel("Status");
  JTextField  tStatus = new  JTextField("0");
  JTextField  tStatusBin = new  JTextField("0");

  JLabel  lSetBit = new JLabel("Bit setzen");
  JSpinner spSetBit;
  JButton bnSetBit = new JButton("Bit setzen");

  JLabel  lDelBit = new JLabel("Bit löschen");
  JComboBox<String> cbDelBit;
  JButton bnDelBit = new JButton("Bit löschen");


  JLabel  lCheckBit = new JLabel("Bit's abfragen");
  JList<String> jCheckListBit;
  JButton bnCheckBit = new JButton("Bit's abfragen");

//   BitValue bitValue = new BitValue(0);


  public Aufgabe03() {
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    setGUI();
  }

  private void setGUI() {
    setSize(700, 490);
    setTitle("Hausaufgabe 03");

  this.getContentPane().setLayout( new GridBagLayout() );

    this.getContentPane().add(lStatus,
         new GridBagConstraints(
             0, 0, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
            GridBagConstraints.NONE,
            new Insets(20,20,20,20), 0, 0));

    this.getContentPane().add(tStatus,
         new GridBagConstraints(
             1, 0, 1, 1, 50.0, 0.0
            ,GridBagConstraints.WEST,
            GridBagConstraints.HORIZONTAL,
            new Insets(20,0,20,0), 0, 0));

    this.getContentPane().add(tStatusBin,
         new GridBagConstraints(
             2, 0, 1, 1, 50.0, 0.0
            ,GridBagConstraints.EAST,
            GridBagConstraints.HORIZONTAL,
            new Insets(20,20,20,20), 0, 0));
```

```
SpinnerNumberModel model1 = new SpinnerNumberModel( 0, 0, 31, 1 );
spSetBit = new JSpinner( model1 );


this.getContentPane().add(lSetBit,
     new GridBagConstraints(
         0, 1, 1, 1, 0.0, 0.0
        ,GridBagConstraints.WEST,
        GridBagConstraints.NONE,
        new Insets(20,20,20,20), 0, 0));

this.getContentPane().add(spSetBit,
     new GridBagConstraints(
         1, 1, 1, 1, 100.0, 0.0
        ,GridBagConstraints.WEST,
        GridBagConstraints.HORIZONTAL,
        new Insets(20,0,20,0), 0, 0));

this.getContentPane().add(bnSetBit,
     new GridBagConstraints(
         2, 1, 1, 1, 0.0, 0.0
        ,GridBagConstraints.EAST,
        GridBagConstraints.NONE,
        new Insets(20,20,20,20), 0, 0));


      bnSetBit.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          bnSetBit_Click();
        }
      });


String[] items = new String[32];
for (int i=0; i<items.length; i++) {
  items[i] = i+". Bit";
}

cbDelBit= new JComboBox<String>(items);


this.getContentPane().add(lDelBit,
     new GridBagConstraints(
         0, 2, 1, 1, 0.0, 0.0
        ,GridBagConstraints.WEST,
        GridBagConstraints.NONE,
        new Insets(20,20,20,20), 0, 0));

this.getContentPane().add(cbDelBit,
     new GridBagConstraints(
         1, 2, 1, 1, 100.0, 0.0
        ,GridBagConstraints.WEST,
        GridBagConstraints.HORIZONTAL,
        new Insets(20,0,20,0), 0, 0));

this.getContentPane().add(bnDelBit,
     new GridBagConstraints(
         2, 2, 1, 1, 0.0, 0.0
        ,GridBagConstraints.EAST,
        GridBagConstraints.NONE,
        new Insets(20,20,20,20), 0, 0));


      bnDelBit.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          bnDelBit_Click();
        }
      });
```

```
  jCheckListBit= new JList<String>(items);


  this.getContentPane().add(lCheckBit,
       new GridBagConstraints(
           0, 3, 1, 1, 0.0, 0.0
          ,GridBagConstraints.NORTHWEST,
          GridBagConstraints.NONE,
          new Insets(20,20,20,20), 0, 0));

  this.getContentPane().add( new JScrollPane(jCheckListBit),
       new GridBagConstraints(
           1, 3, 1, 1, 100.0, 0.0
          ,GridBagConstraints.WEST,
          GridBagConstraints.HORIZONTAL,
          new Insets(20,0,20,0), 0, 0));

  this.getContentPane().add(bnCheckBit,
       new GridBagConstraints(
           2, 3, 1, 1, 0.0, 0.0
          ,GridBagConstraints.NORTHEAST,
          GridBagConstraints.NONE,
          new Insets(20,20,20,20), 0, 0));


        bnCheckBit.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            bnCheckBit_Click();
          }
        });



        setFont( this.getContentPane() );
         // wg. JScrollPane
        jCheckListBit.setFont( new Font("Arial", Font.BOLD,18)  );

} // setGUI


private void setFont(Container parent) {
  //System.out.println("parent: "+parent+"      "+parent.getComponentCount()  );
  for (int i=0; i<parent.getComponentCount(); i++ ) {
     Component c = parent.getComponent(i);
     if (c instanceof JPanel ) {
        JPanel  pn = (JPanel ) c;
        setFont(pn);
     }
     else if (c instanceof JScrollPane ) {
        JScrollPane  sc = (JScrollPane ) c;
        setFont(sc);
     }
      else {
        c.setFont( new Font("Arial", Font.BOLD,18)  );
     }
  }
}


private void bnSetBit_Click() {
  // hier fehlt Code
}

private void bnDelBit_Click() {
  // hier fehlt Code
}
```

```
   public void MessageBox( String sTitle, String sMessage) {
        JOptionPane.showConfirmDialog(null,sMessage,sTitle,
        JOptionPane.DEFAULT_OPTION,
        JOptionPane.INFORMATION_MESSAGE);  // Symbol
   } // MessageBox

  private void bnCheckBit_Click() {
    // hier fehlt Code
    MessageBox( "Check von Bits", "Alle Bits sind gesetzt");
    MessageBox( "Check von Bits", "Nicht alle Bits sind gesetzt");
  }

  public static void main(String[] args) {
    Aufgabe03 frame = new Aufgabe03();
    frame.setVisible(true);
 }

}  // Aufgabe03
```

3. Teilaufaufgaben: Event-Methoden implementieren

Die ersten beiden Schalter wandeln die Bits der Instanz „bitValue um und geben den neuen Wert in den beiden Status-JTextField aus. Im linken JTextField wird der Wert als dezimaler String ausgegeben. Im rechten Status-JTextField **muss** der Wert in Binärdarstellung ausgegeben werden. Der letzte Schalter prüft, ob **alle** Bits gesetzt sind. Die Check-Methoden gibt als Rückgabewert true oder false zurück. Je nach Wert muss mit Hilfe der Methode „MessageBox“ eine Nachricht ausgegeben werden.

Schalter-Aktionen:

- Bit setzen
  - Das Bit des JSpinners, eine Zahl zwischen 0 und 31, soll in der Instanz bitValue gesetzt werden.
  - Dazu muss der Wert des JSpinners in einen long-Wert umgewandelt werden.
  - Hier wird noch eine Umrechnung benötigt:
    - Aus 0 wird 1
    - Aus 1 wird 2
    - Aus 2 wird 4
    - Aus 3 wird 8
    - Denkbar wäre eine Funktion à la „>>“ oder eine Schleife
  - Danach wird die Methode „setBit“ aufgerufen.
  - Am Schluss sollen die beiden Status-JTextFields neu gesetzt werden.
- Bit löschen
  - Das Bit der ComboBox soll in der Instanz bitValue gelöscht werden.
  - Dazu muss der Index der ComboBox in einen long-Wert gespeichert werden.
  - Hier wird noch eine Umrechnung benötigt:
    - Aus 0 wird 1
    - Aus 1 wird 2
    - Aus 2 wird 4
    - Aus 3 wird 8
    - Denkbar wäre eine Funktion à la „>>“oder eine Schleife
  - Danach wird die Methode „delBit“ aufgerufen.
  - Am Schluss sollen die beiden Status-JTextFields neu gesetzt werden.
- Bits abfragen
  - Die markierten Bits der JList sollen in der Instanz bitValue abgefragt werden.
  - In einer Schleife sollten die Bits der JList abgefragt werden und eine lokale Variable gesetzt werden.
  - Hier wird noch eine Umrechnung benötigt:
    - Aus 0 wird 1
    - Aus 1 wird 2
    - Aus 2 wird 4
    - Aus 3 wird 8
    - Denkbar wäre eine Funktion à la „>>“oder eine Schleife
  -
  - Die Methode „checkBits“ prüft dann die Bits.
  - Das Ergebnis soll mittels MessageBox ausgegeben werden. Jeweils für den then und else-Fall.

Bilder der Musterlösung:

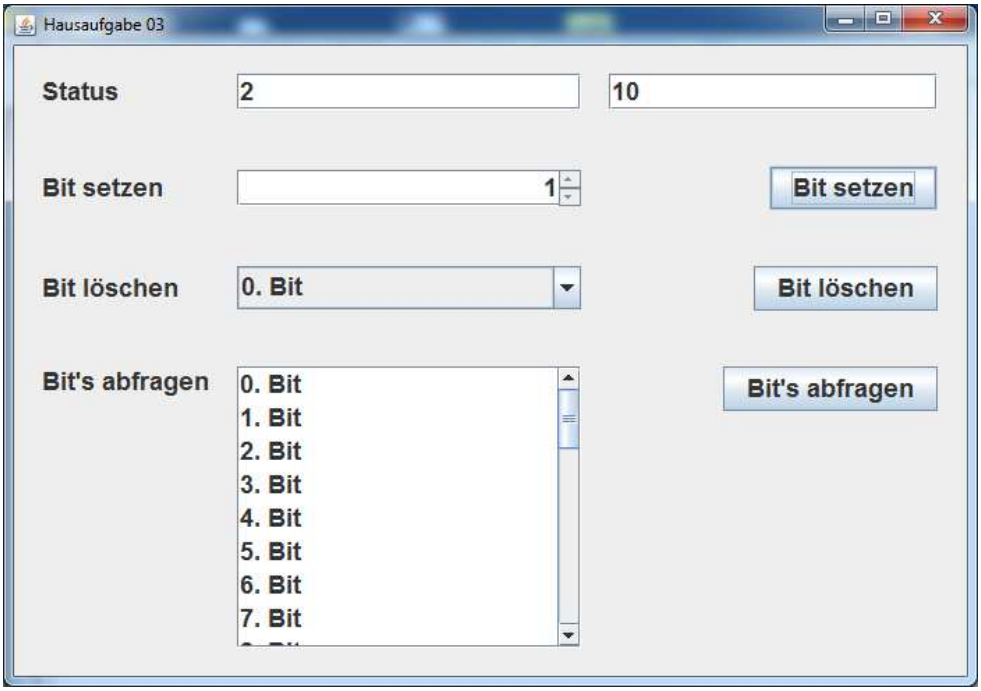

Abbildung 2 Bit 1 setzen

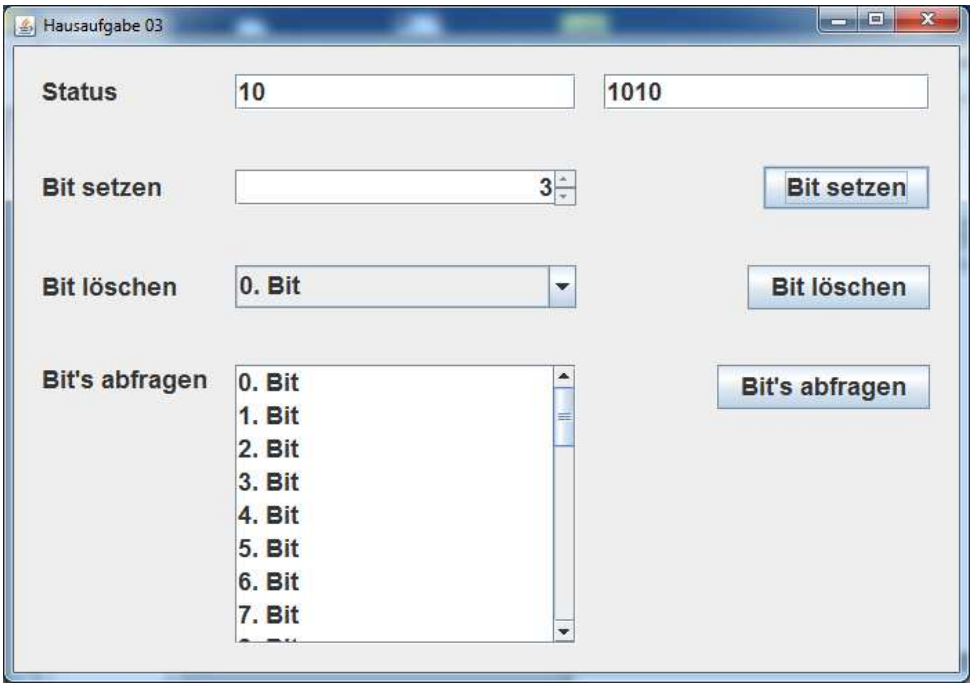

Abbildung 3 Bit 3 setzen

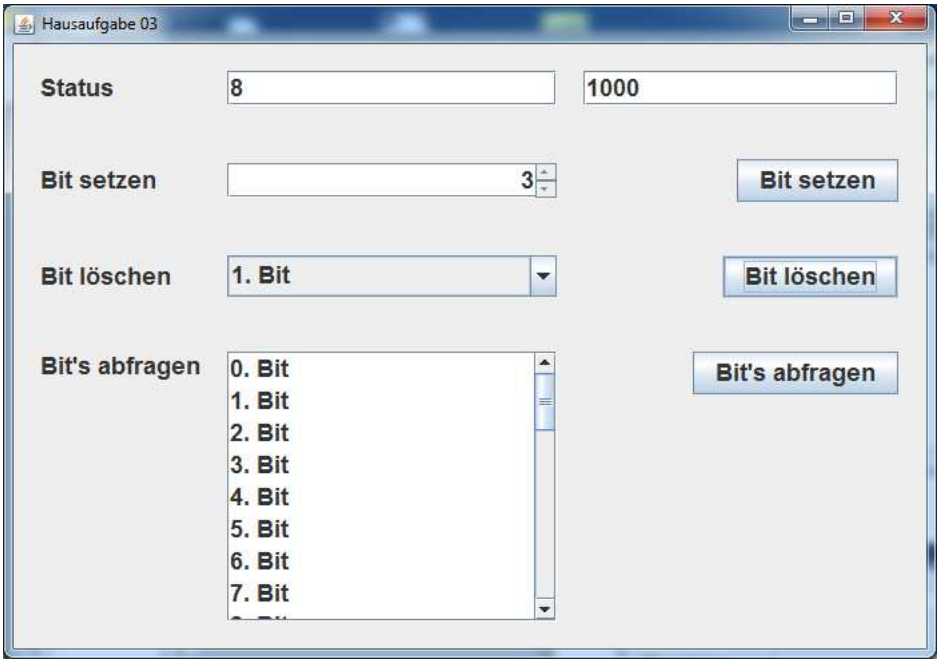

Abbildung 4 Bit 1 löschen

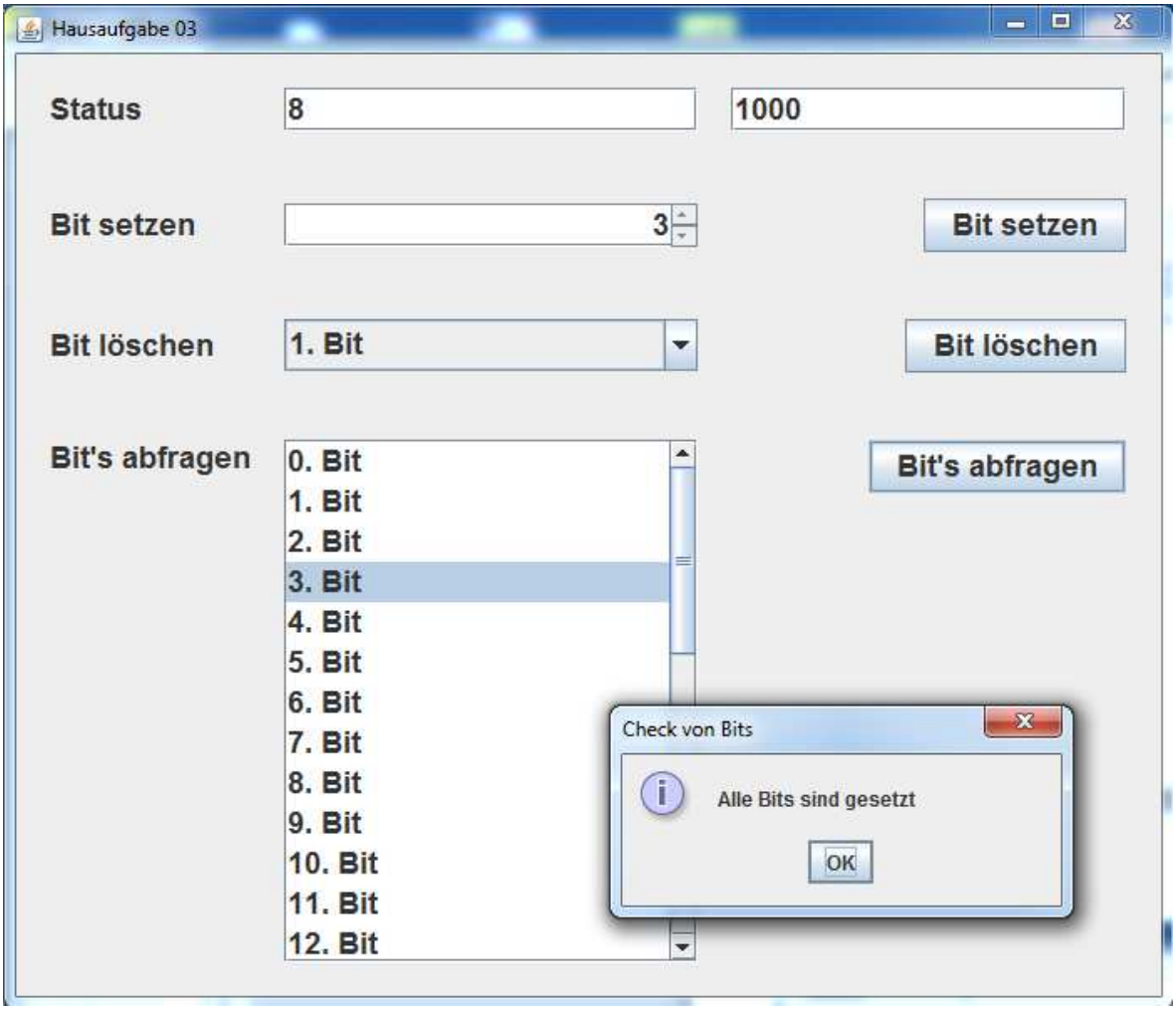

Abbildung 5 Bit 1 abfragen

Beispiele für die Funktion „Bit setzen“:

Ausgangpunkt: Dezimalwert	Bit setzen	Ergebnis
0	2	4
2	2	6
4	2	4
0	3	8
0	10	1024
1	10	1025

Beispiele für die Funktion „Bit löschen“:

Ausgangpunkt: Dezimalwert	Bit löschen	Ergebnis
0	2	0
2	2	2
4	2	0
8	3	0
1025	10	1
1025	1	1024

Beispiele für die Funktion „Bit abfragen“:

Ausgangpunkt: Dezimalwert	Bit`s abfragen	Ergebnis (wahr/falsch)
0	2	falsch
255	1,3,5	Wahr
255	8	Falsch
1025	1,10	Wahr
1024	1,10	Falsch
1025	1,2,10	Falsch

Mathematik

Bitweise Oder-Funktion:

ODER	0	1
0	0	1
1	1	1

Die Bit-Funktion läuft in Java über alle einzelne Bits:

Java:
```
int i=22;
int j=13;
int k;

k = i | j;
```

$22_{10} = 0010110_2$
$13_{10} = 0001101_2$

Ergebnis:
$31_{10} = 0011111_2$

Bitweise Und-Funktion:

UND	0	1
0	0	0
1	0	1

Die Bit-Funktion läuft in Java über alle einzelne Bits:

Java:
```
int i=22;
int j=13;
int k;

k = i & j;
```

$22_{10} = 0010110_2$
$13_{10} = 0001101_2$

Ergebnis:
$04_{10} = 0000100_2$

Bitweise Negation-Funktion:

NOT / Negation	Ergebnis
0	1
1	0

Die Bit-Funktion läuft in Java über alle einzelne Bits:

Java:
```
int i=22;
int k;
k = ~i;
```
$22_{10} = 0010110_2$
$= 11111111111111111111111111111111111111111111111111111111111101001_2$

Funktionen

Setzen von Bits:

Die Bits werden mittels des bitweisen Oder-Operator gesetzt.

Java:

```
void test1(int bits) {
    int j=13;
    j = j & bits
}
```

Löschen von Bits:

Die Bits werden mittels des bitweisen Und-Operator gesetzt. Dabei ist aber zu beachten, dass für die Bitmaske alle Bits, die **nicht** beeinflusst werden sollen, auf eins gesetzt werden. Alle Bits, die gelöscht werden sollen, werden auf Null gesetzt. Da man unabhängig von der Länge des Datentyp programmieren soll, benutzt man den NOT-Operator.

Java:

```
void test2(int bits) {
    int j=13;
    int k = ~bits;          // Negation der „bits“
    j = j & k
}
```

Abfragen von Bits:

Das Abfragen der Bits geschieht mittels des UND-Operators. Dabei werden die Suchbits im Gegensatz zum Löschen, nicht verändert. Die Ergebnisse können vielfältig sein. Eigentlich 2 hoch „Anzahl der Bits“.

Beispiele:

Abfragt wird das 1. Bit. Man zählt von Null.
Bei der Abfrage eines Bits gibt es als Ergebnis nur Null oder den Bit-Wert

Interner Wert: 118

7. Bit	6	5	4	3	2	1. Bit	0. Bit
0	1	1	1	0	1	1	0

Suchmaske: 2

7. Bit	6	5	4	3	2	1. Bit	0. Bit
0	0	0	0	0	0	1	0

Ergebnis: 2

7. Bit	6	5	4	3	2	1. Bit	0. Bit	
0	0	0	0	0	0	**1**	0	
128	64	32	16	8	4	2	1	Wertigkeit der Bits

Interner Wert: 116

7. Bit	6	5	4	3	2	1. Bit	0. Bit
0	1	1	1	0	1	0	0

Suchmaske

7. Bit	6	5	4	3	2	1. Bit	0. Bit
0	0	0	0	0	0	1	0

Ergebis 0

7. Bit	6	5	4	3	2	1. Bit	0. Bit
0	0	0	0	0	0	~~1~~ 0	0

Abfragt wird das 1. und 3. Bit. Man zählt von Null.
Bei der Abfrage von zwei Bits gibt es als Ergebnis vier Möglichkeiten:

0 0 0
0 1 2
1 0 8
1 1 10

Interner Wert: 118

7. Bit	6	5	4	3	2	1. Bit	0. Bit
0	1	1	1	0	1	1	0

Suchmaske: 10

7. Bit	6	5	4	3	2	1. Bit	0. Bit
0	0	0	0	**1**	0	1	0

Ergebnis: 2

7. Bit	6	5	4	3	2	1. Bit	0. Bit	
0	0	0	0	0	0	**1**	0	
128	64	32	16	8	4	2	1	Wertigkeit der Bits

In diesem Fall muss man sich überlegen, ob man prüfen will, ob **alle** Bits gesetzt sind. Dann muss das Ergebnis gleich der Suchmaske sein. Will man nur prüfen, ob **ein** Bit gesetzt ist, prüft man, ob das Ergebnis größer Null ist.

Java:

```
void test3(int bits) {
    int j=13;
    int j &= bits;

    if(j == bits){
        Syso(„Bits sind gesetzt“);
    } else {
        Syso(„Nicht alle Bits sind gesetzt :-( “);
    }
}

// oder so

boolean bitGesetzt(int internerWert, int suchMaske) {
    int ergebnis =  internerWert & suchMaske;

    if(ergebnis == suchMaske){
        return true;
    } else {
        return false;
    }
}
```

Anhang:

Klasse JSpinner:

Folgender Programmausschnitt zeigt mehrere Arten von JSpinner an.

```
JSpinner spInteger;
                                        // Aktueller Wert, Min, Max, Increment
spInteger = new JSpinner( new SpinnerNumberModel(0,  0, 100, 1)   );
```

```
JSpinner spDouble;
                                        // Aktueller Wert, Min, Max, Increment
spDouble = new JSpinner( new SpinnerNumberModel(5.5,  0, 100, 0.5)   );
```

setzen eines Wertes in einem JSpinner:

```
SpinnerInt.setValue(  new Integer( 123 )  );
SpinnerDouble.setValue(  new Double ( 55.6 )  );
```

holen eines Wertes aus einen Integer-JSpinner:

```
Integer I = (Integer) Spinner1.getValue(   );
int k = I.intValue();
```

oder

```
int k = ( (Integer) Spinner1.getValue() ).intValue();
```

Klasse JComboBox:

```
public JComboBox ();                    // leere Liste
public JComboBox (Object [] liststd);   // Array mit Objekten
public JComboBox (Vector liststd);      // Vector mit Objekten
```

setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
Nur ein Wert darf selektiert werden

boolean isSelectionEmpty()
~~Wurde mindestens ein Wert selektiert~~
True: Es wurde kein Element gewählt, False: Es wurde ein Element gewählt

int getSelectedIndex()
Index des selektierten Eintrags

isSelectedIndex(int)

Object getSelectedValue()
Gibt das aktuelle Objekt zurück.
Falls kein Eintrag ausgewählt wurde, gibt es null zurück.

```
void   setListData(Object[] listData)        // Update der Einträge
void   setListData(Vector listData)          // Update der Einträge
```

Klasse JList:

```
public JList ();              // leere Liste
public JList (Object [] liststd);  // Array mit Objekten
public JList (Vector liststd);     // Vector mit Objekten
```

setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
Nur ein Wert darf selektiert werden

setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
Ein Wert selektiert, oder ein Bereich

setSelectionMode(ListSelectionModel. MULTIPLE_INTERVAL_SELECTION);
Beliebiger Bereich darf selektiert werden

boolean isSelectionEmpty()
~~Wurde mindestens ein Wert selektiert~~
True: Es wurde kein Bereich oder Element gewählt, False: Es wurde ein Bereich oder Element gewählt

int getSelectedIndex()
Index des selektierten Eintrags

int[] getSelectedIndices()
~~Index~~ aller selektierten Einträge
Indizes aller selektierten Einträge

isSelectedIndex(int)

Object getSelectedValue()
Gibt das aktuelle Objekt zurück.
Falls kein Eintrag ausgewählt wurde, gibt es null zurück.

Object[] getSelectedValues()
Rückgabe aller selektierten Objekte oder null.

```
void   setListData(Object[] listData)        // Update der Einträge
void   setListData(Vector listData)          // Update der Einträge
```