JAVA und die grafische Benutzeroberfläche

Events, Tastaturereignisse

Alfred Nussbaumer

JAVA stellt mit dem AWT (*Abstract Window Toolkit*), mit Swing (JFC, *Java Foundation Classes*) oder SWT (*Standard Widget Toolkit*, IBM) Klassenbibliotheken für GUI-Anwendungen zur Verfügung. Die wichtigsten Entwicklungsumgebungen dazu sind die beiden OpenSource-Projekte Eclipse (IBM) und Netbeans (SUN). Beide Entwicklungsumgebungen sind selbst in JAVA geschrieben und können für Windows und Linux frei bezogen werden (vgl. **[1]**, **[2]**).

Für das GUI (*Graphical User Interface*) sind im Gegensatz zur Textkonsole Objekte wie das Anwendungsfenster selbst (*Window*), Ein- und Ausgabeobjekte wie *Labels*, *Buttons*, oder *Scrollbars* sowie Methoden nötig, die Interaktionen (*Events*) mit den grafischen Objekten zulassen. Zu diesen zählen wir beispielsweise *KeyEvents*, *MouseEvents* oder *WindowEvents*.

Dieser Beitrag soll eine kurze allgemeine Einführung zu Ereignissen enthalten. In einigen Beispielen soll schließlich gezeigt werden, wie Tastaturereignisse verarbeitet werden können.

1. Ereignisbehandlung

Bei einer GUI-Anwendung können Ereignisse an verschiedenen Orten des Bildschirms, zu verschiedenen Zeiten von verschiedenen Objekten ausgelöst werden. Um auf diese so genannten *Events* reagieren zu können, sind in JAVA verschiedene Interfaces (*EventListeners*) definiert, die zum jeweiligen Ereignis passende Methoden enthalten. Häufig verwendete Ereignisklassen sind:

- `WindowEvent` – wird in Zusammenhang mit dem Anwendungsfenster ausgelöst, beispielsweise beim Schließen des Fensters.
- `ActionEvent` – wird beispielsweise beim Klick auf eine Schaltfläche ausgelöst.
- `KeyEvent` – wird über die Tastatur ausgelöst.
- `MouseEvent` – wird durch Mausbewegung oder durch das Drücken einer Maustaste ausgelöst.

Tritt ein bestimmtes Ereignis auf, so reagieren entsprechende Methoden, indem sie einerseits das Ereignis entgegen nehmen und andererseits gewünschte Reaktionen ausführen. Diese Aktionen hängen im Allgemeinen von der Anwendung ab und werden durch den Programmierer festgelegt. Die Schnittstellen (*Listeners*) tragen aussagekräftige Namen; die wichtigsten sind:

- `ActionListener`
  - `actionPerformed(Actionevent e)`
- `WindowListener`
  - `windowActivated(WindowEvent e),`
  - `windowClosed(WindowEvent e),`
  - `windowClosing(WindowEvent e),`
  - `windowDeactivated(WindowEvent e),`
  - `windowDeiconified(WindowEvent e),`
  - `windowIconified(WindowEvent e),`
  - `windowOpened(WindowEvent e)`
- `KeyListener`
  - `keyPressed(KeyEvent e)`
  - `keyReleased(KeyEvent e)`
  - `keyTyped(KeyEvent e)`
- `MouseListener`
  - `mouseClicked(MouseEvent e)`
  - `mouseEntered(MouseEvent e)`
  - `mouseExited(MouseEvent e)`
  - `mousePressed(MouseEvent e)`
  - `mouseReleased(MouseEvent e)`
- `MouseMotionListener`
  - `mouseDragged(MouseEvent e)`
  - `mouseMoved(MouseEvent e)`

Eine detaillierte Beschreibung ist in der JAVA-Dokumentation **([4])** gegeben. Beispiele finden sich u. A. in den Tutorials der Firma SUN (vgl. **[3]**).

Soll die Ereignisbehandlung für eine JAVA-Klasse implementiert werden, so ist es zunächst notwendig, den entsprechenden *Listener* zu erweitern. Im Programmcode müssen dann alle mit dem *Listener* verbundenen Methoden deklariert werden. Zuletzt müssen die *Listener* mit den vorgesehenen Objekten verbunden werden (z.B. ein `WindowListener` an ein Window-Objekt, oder ein `ActionListener` an ein Schaltflächen-Objekt usf.). In den nächsten Beispielen soll diese Vorgangsweise im Detail vorgestellt werden.

2. Tastaturereignisse (Keyboard Events)

Für Tastaturereignisse verwendet man ein `KeyListener`-Interface. Mit der Methode `addKeyListener(this)` wird festgelegt, dass alle Tastaturereignisse in Zusammenhang mit dem Applet entgegen genommen werden sollen. Das `KeyListener`-Interface enthält die Methoden `keyPressed()`, `keyReleased()` und `keyTyped()`. Sie müssen innerhalb der Anwendung implementiert werden.

Im folgenden Beispiel soll ein Quadrat mit den Pfeiltasten über den Bildschirm bewegt werden. Dazu wird nach jedem Drücken einer Cursortaste die neue Position des Quadrates bestimmt und anschließend der ganze Bildschirmbereich neu ausgegeben. Sollte das Quadrat den Rand des Bereiches überschreiten, setzt man es an den Rand vis-a-vis – das Quadrat bewegt sich sozusagen auf einem Torus.

Programmcode Move.java:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Move extends java.applet.Applet
              implements KeyListener {
```

```
  private int xpos = 25;
  private int ypos = 15;
  private Color ly = new Color(255,255,245);

  public void init() {
    addKeyListener(this);
  }

  public void keyPressed(KeyEvent e) {
    switch (e.getKeyCode()) {
      case 37: xpos--; break;
      case 38: ypos--; break;
      case 39: xpos++; break;
      case 40: ypos++; break;
      default:
    }
    if (xpos < 0) xpos = 49;
    if (ypos < 0) ypos = 29;
    if (xpos > 49) xpos = 0;
    if (ypos > 29) ypos = 0;
    repaint();
  }

  public void keyReleased(KeyEvent e) {
  }

  public void keyTyped(KeyEvent e) {
  }

  public void paint (Graphics g) {
    g.setColor(ly);
    g.fillRect(0,0, 500, 300);
    g.setColor(Color.green);
    g.fillRect(xpos*10, ypos*10, 8,8);
  }
}
```

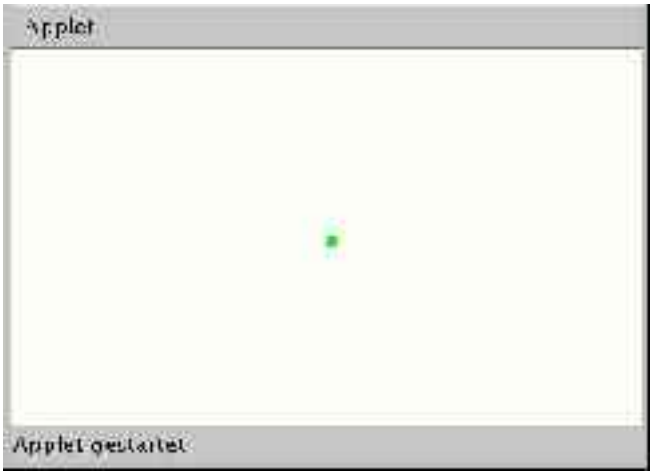


Abb. 1: Applet „move“ - ein grünes Quadrat wird mit den Pfeiltasten über den Bildschirm gesteuert.

Damit nicht immer alle Methoden eines *Listeners* implementiert werden müssen, verwendet man so genannte **Adapterklassen**, die leere Definitionen aller zugehöriger Methoden enthalten. Auf diese Weise genügt es, nur die jeweils verwendeten Methoden zu kodieren. Im folgenden Beispiel wird ein `KeyListener` an das Komponentenobjekt gebunden, das mit dem Öffnen des Applet-Fensters vorhanden ist.

```
  public void init() {
    this.addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent e) {
        ...
      }
    });
  }
```

Ist das Applet aktiv und wird eine Taste gedrückt, so wird ein Tastaturereignis vom Typ `KeyEvent` ausgelöst. Daraufhin wird die KeyListener-Methode `keyPressed()` ausgeführt: Bei ihrer Implementation durch den Programmierer enthält sie entweder alle Angaben, was mit dem Tastaturereignis passieren soll, oder sie übergibt das Ereignis an eine weitere Methode.

Tastaturereignisse können einfache Spiele steuern: Spielfiguren können beispielsweise mit den Pfeiltasten bewegt werden, oder 3x3-Eingabefelder werden über den Ziffernblock der Tastatur angesteuert. Aus der Dokumentation können die Konstanten nachgelesen werden, die den jeweiligen Tastaturcodes zugeordnet sind. In den folgenden Beispielen liefert die `KeyEvent`-Methode `getKeyCode()` den Tastaturcode in Form einer ganzen Zahl. Diese wird in einer Mehrfach-Verzweigung (`switch`-Anweisung) als Selektor verwendet.

Taste	***Tastaturcode***
„Pfeil nach links“	37
„Pfeil nach oben“	38
„Pfeil nach rechts“	39
„Pfeil nach unten“	40
„1“ auf Ziffernblock	97
„2“ auf Ziffernblock	98
...	...
„9“ auf Ziffernblock	105

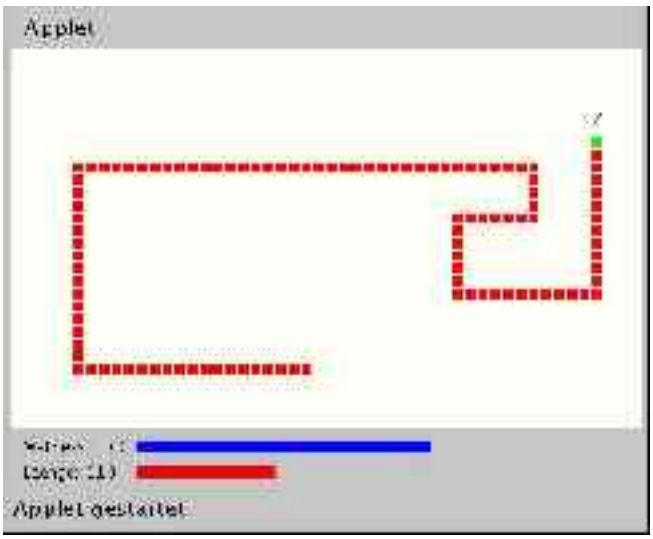
Tabelle 1: Einige Tastaturcodes (Ergebnis der `KeyEvent`-Methode `getKeyCode()`).

3. Ein Klassiker: Das Wurmspiel

Die grundsätzlich einfache Spielidee soll mit einem Applet realisiert werden. Ein Wurm wird mit den Pfeiltasten über ein Spielfeld bewegt. Der Kopf des Wurmes wird beispielsweise durch ein grünes Quadrat, seine Körperglieder durch rote Quadrate angezeigt. Trifft er mit seinem Kopf auf eine Zahl, so frisst er diese, und seine Länge verlängert sich um den gefressenen Zahlenwert. Sobald die Zahl gefressen ist, wird eine Position einer neuen Zahl durch ein zufällig ermitteltes (x/y)-Koordinatenpaar berechnet und an dieser Stelle eine Zufallszahl zwischen 1 und 9 ausgegeben.

Das Kriechen kostet den Wurm Energie. Zur Bewertung werden die Schritte gezählt: Ein einfacher Algorithmus wertet die Wurmlänge und die Anzahl der Schritte aus und bestimmt daraus den Parameter „wellness“; das Spiel endet, wenn der Wurm sich selber „beißt“ oder an den Rand der Zeichenfläche stößt.

Abb. 2: Der Wurm wird um so viele Quadrate länger, wie die



Zahl angibt, die er gerade gefressen hat.

Um die Position der Zufallszahl zu speichern und um zu „wissen“, auf welchem Bereich der Zeichenfläche sich der Wurm bewegt, verwenden wir ein zweidimensionales Array. Seine Dimension ist durch die Anzahl der horizontalen und vertikalen Quadrate gegeben. Alle Datenfelder werden auf den Wert 0 initialisiert. Die Felder, die vom Wurm besetzt sind und ein Wurmglied darstellen, erhalten eine positive Zahl. Die Zufallszahl wird als negativer Wert zwischen -10 und 0 eingetragen.

Die aktuelle Position des Wurmkopfes (des grünen Quadrates) ist in den Koordinaten `xpos` und `ypos` gespeichert. Zusätzlich zu den Koordinaten des Wurmkopfes muss die aktuelle Länge bzw. das Fressergebnis gespeichert werden – dazu dienen die Variablen `laenge` und `frass`.

Applet Wurm.java:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.util.Random;

public class Wurm extends Applet {

  private int k [][] = new int[50][30];
  private int xpos;
  private int ypos;
  private int laenge;
  private int frass;
  private String mitteilung;
  private int counter;

  public wurm() {
    for (int i=0; i<50; i++)
      for (int j=0; j<30; j++) k[i][j] = 0;
    xpos = 25;
    ypos = 15;
    laenge = 1;
    frass = 0;
    mitteilung = " ";
    counter = 1;
  }

  public void init() {
    setzezahl();
    this.addKeyListener(new
       java.awt.event.KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                this_keyPressed(e);
       }
    });
  }

  public void setzezahl() {
    Random r = new Random();
    int z=0;
    int xposr;
    int yposr;
    while (z==0) {
      xposr = r.nextInt(48)+1;
      yposr = r.nextInt(28)+1;
      if (k[xposr][yposr] == 0) {
        k[xposr][yposr]=(r.nextInt(9)+1)*(-1);
        z++;
      }
    }
  }

  public void this_keyPressed (KeyEvent e) {
    switch (e.getKeyCode()) {
      case 37: xpos--; break;
      case 38: ypos--; break;
      case 39: xpos++; break;
      case 40: ypos++; break;
      case 32: break;
      default:
    }
    if ((xpos >=0) && (ypos >= 0) && (xpos < 50)
              && (ypos < 30)) check();
    bewege();
    repaint();
  }

  public void check() {
    if (k[xpos][ypos]<0) {
      frass+=(k[xpos][ypos]*(-1));
      for (int i=0;i<50;i++)
        for (int j=0;j<30;j++)
          if (k[i][j]>0) k[i][j]+=frass;
      laenge+=frass;
      setzezahl();
    }
    if (k[xpos][ypos]>0) ende();
  }

  public void bewege()  {
    counter++;
    if ((xpos<0) || (ypos<0) || (xpos>=50) ||
       (ypos>=30)) ende();
    else {
      k[xpos][ypos]=laenge+1;
      for (int i=0;i<50;i++)
        for (int j=0; j<30;j++)
          if (k[i][j]>0) k[i][j]--;
      if (frass > 0) frass--;
    }
  }

  public void ende() {
    mitteilung = "und aus!";
    for (int i=0; i<50; i++)
      for (int j=0; j<30;j++) k[i][j]=1000;
  }

  public void paint (Graphics bs) {
    Color ly = new Color(255,255,245);
```

```
    bs.setColor(ly);
    bs.fillRect(0,0, 500, 300);
    for (int i=0; i<50; i++) {
      for (int j=0; j<30; j++) {
        if (k[i][j]>0) {
          bs.setColor(Color.red);
          bs.fillRect(i*10,j*10, 8,8);
        }
        if ((k[i][j]>-10) && (k[i][j]<0)) {
          bs.setColor(Color.blue);
          bs.drawString(" " + (k[i][j]*(-1)),
                                i*10-4, j*10+9);
        }
      }
    }
    bs.setColor(Color.green);
    bs.fillRect(xpos*10,ypos*10, 8,8);
    bs.setColor(Color.black);
    bs.drawString("Wellness: " + (int) ((((double)
laenge-1) / (double) counter)*1000), 10, 320);
    bs.drawString("Laenge: " + laenge, 10,340);
    bs.drawString(mitteilung, 300, 340);
    bs.setColor(Color.blue);
    bs.fillRect(100, 310, (int) ((((double)
laenge-1) / (double) counter) * 1000), 10);
    bs.setColor(Color.red);
    bs.fillRect(100, 330, laenge, 10);
  }
}
```

Die Methode `setzezahl()` dient zum Setzen einer Zufallszahl: Zufällig wird eine Position auf der Zeichenfläche ausgewählt. Ist dort kein Wert eingetragen (d.h. befindet sich dort keine Teil des Wurmes), so wird eine (negative) Zahl zwischen -10 und -1 eingetragen, und die While-Schleife bricht ab.

Besondere Beachtung verdienen die Methoden `check()` und `bewege()`. Die Methode `check()` wird nach jedem Tastendruck aufgerufen, wenn der Wurm die Zeichenfläche nicht verlassen hat. Zunächst wird überprüft, ob der Wurmkopf eine Zahl zum Fressen gefunden hat. Dies ist der Fall, wenn der Eintrag `k[xpos][ypos]` negativ ist: Der Eintrag wird mit (-1) multipliziert und in die Variable `frass` übernommen. Außerdem werden alle Einträge `k[i][j]` (sie stellen ja den Wurmkörper dar) um den Wert der Variablen frass erhöht. Abschließend wird eine neue Zufallszahl ausgegeben.

In der Methode `bewege()` wird zunächst die aktuelle Wurmlänge vermehrt um 1 an der aktuellen Position des Wurmkopfes eingetragen und anschließend jeder Wert, der ein Körperglied des Wurmes repräsentiert um 1 vermindert. Auf diese Weise erhält das Körperglied, das unmittelbar auf den Kopf folgt, den Wert `laenge - 1`, das nächste den Wert `laenge - 1`, usf. bis zum letzten Körperglied mit dem Wert `1`. Dieses erhält bei der nächsten Bewegung den Wert `0` und repräsentiert daher keinen Wurmteil mehr: Der Wurm ist weitergekrochen...

Die Methode `ende()` wird aufgerufen, wenn die Position des Wurmkopfes außerhalb des Spielfeldes zu liegen käme, oder wenn der Eintrag im Datenarray `k` positiv ist und somit einen Wurmkörperteil repräsentiert; in diesem Fall hätte der Wurm sich selbst gebissen.

4. Aufgaben, Ausblick

Die Dateneingabe über die Tastatur bei Taschenrechner und Handy motiviert eine Vielzahl von „Tastaturspielen“. Je nach Spielidee kommen hier ausgefeilte Algorithmen zum Einsatz.

1. Das erste Beispiel „Move.java“ soll so erweitert werden, dass mit Hilfe des Quadrates Zufallszahlen an Zufallspositionen des Bildschirmes gesammelt werden sollen. Bei jedem Treffer wird die Anzahl der Barrieren am Bildschirm erhöht: Das Spiel endet, wenn das Quadrat auf eine Barriere trifft.

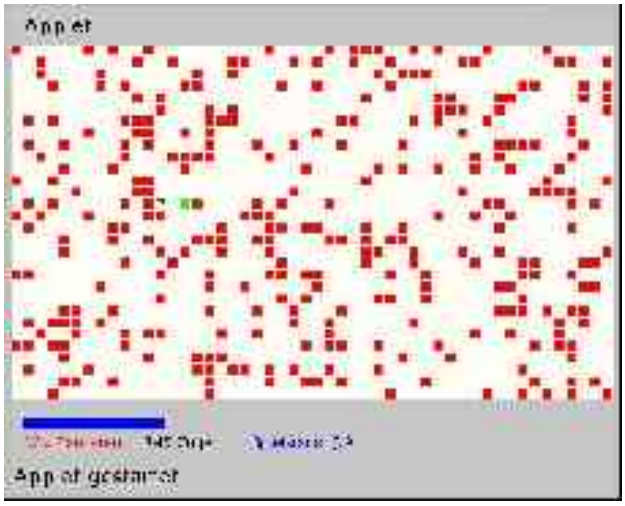


Abb. 3: Die Anzahl der Barrieren nimmt nach jedem gesammelten Gutpunkt zu...

2. Das Wurmspiel „Wurm.java“ ist so um Barrieren zu erweitern, dass sich je nach Spielstand immer schwierigere Situationen ergeben.

3. Bei einem einfachen („klassischen“) Spiel, das auf dem Ziffernblock gespielt wurde, geht es darum, 9 Quadrate umzufärben, die den Ziffern 1 – 9 zugeordnet sind. Einige anfänglich zufällig ausgewählte Quadrate sind eingefärbt. Die Zifferntasten 1, 3, 7 und 9 färben das zugeordnete Quadrat sowie die drei anliegenden Quadrate um; die Zifferntasten 2, 4, 6 und 8 färben die drei Quadrate der zugehörigen Quadratseite um, und die Zifferntaste 5 färbt alle Quadrate um. Ziel ist es, alle Quadrate in möglichst wenigen Spielzügen auf weiß zu färben... (vgl **[7]**).

Abb. 4: „Invert“ lädt zum Knobeln ein: 9 Quadrate sollen mit Hilfe der Zifferntasten umgefärbt werden.

4. „Tic-Tac-Toe“ soll über den Ziffernblock der Tastatur gegen den PC gespielt werden. Nach welchem Algorithmus soll der PC setzen? Wie wird überprüft, wer gewonnen hat?

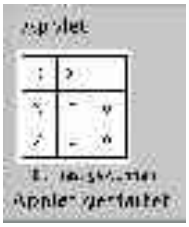


Abb. 5: Wer spielt „Tic-Tac-Toe“ gegen den PC?

5. Ein elementares Trainingsprogramm zum Anwenden des „10-Finger-Systems“ auf der Computertastatur ist zu entwerfen. Nach der Ausgabe eines zufällig ausgewählten Buchstabens wird der Tastaturcode der nächsten Eingabe überprüft und ausgewertet.

Tastaturereignisse sind nur ein kleiner Teil möglicher Ereignisse, die im Rahmen eines GUI auftreten. Einige häufig auftretende Ereignisse wie der Klick auf eine Schaltfläche, Bewegen oder Klicken mit der Maus oder das Schließen eines Anwendungsfensters werden in ihren Grundzügen und in Beispielen in einem späteren JAVA-Beitrag erläutert.

5. Literatur, Weblinks

[1] http://www.eclipse.org (Informationen und Download von Eclipse)

[2] http://www.netbeans.org/ (Informationen und Download von Netbeans)

[3] http://java.sun.com/docs/books/tutorial/uiswing/ (Tutorial für Swing-Komponenten)

[4] http://java.sun.com/j2se/1.4.2/docs/index.html (Dokumentation aller verfügbaren Packages)

[5] Herbert Schildt, „Java 2 Ent-Packt“, mitp-Verlag

[6] Christian Ullenboom, „Java ist auch eine Insel“, Galileo Computing

[7] http://www.gymmelk.ac.at/nus/informatik/wpf/JAVA/ (Unterrichtsbeispiele zu JAVA)