1.Grafik GUI-Programmierung 2
1.1.Ziele 2
1.2.Projekte 2
1.3.Elemente 2
1.3.1.JFrame, JPanel 2
1.3.2.Layouts 3
1.3.3.JLabel 3
1.3.4.JButton 3
1.3.5.Das grafische Koo 3
1.3.6.Die "Leinwand" und die Methode paint() 4
1.4.Grafikfunktionen 5
1.4.1.Ein Fenster: GrafikDemo.java 5
1.4.2.Linien 7
1.4.3.Rechtecke 8
1.4.4.Polygon 9
1.4.5.Kreis 10
1.5.Farben 10
1.6.Erste Beispiele 10
1.6.1.+Aufgabe: DemoGrafik.java 11
1.6.2.Aufgabe: SwingQuadrat.java 12
1.6.3.Aufgabe: VisualSort.java 12
1.6.4.Aufgabe: SwingBildlaufSpiel.java 13
1.6.5.+Aufgabe: SwingSpirale 13
1.6.6.+Aufgabe: SwingLabyrinth 15
1.6.7.+Aufgabe: SwingDatumZeit.java 15
1.6.8.Hinweis: Datum, Zeit 16
1.6.9.+Hinweis: AnalogUhr 16
1.7.+Ein erstes Spiel: SpielFirstGame.java 17
1.7.1.Der Spieler 18
1.7.2.Der Timer 18
1.7.3.public void paint(java.awt.Graphics g) 18
1.7.4.Animation mit Timer und Bildschirmdarstellung 18
1.7.5.Tastatureingaben, Cursortasten 19
1.8.+Bildschirm Zwischenspeicherung 21
1.8.1.+Aufgabe: SpielFirstGame 23
1.8.2.+Aufgabe: SpielSnake 23
1.8.3.+Aufgabe: SpielTetris 23
1.8.4.+Aufgabe: SpielPacman 23
1.9.Zusammenfassung 23
1.10.Ausblick 23

1. Grafik GUI-Programmierung

Die in diesem Kapitel Software wurde mittels Eclipse Neon.2 (4.6.2) / WindowBuilder 1.9.0 getestet.

1.1. Ziele

☑ Einfache GUI mit Eclipse-Windowbuilder erstellen können

☑ Vorkenntnisse: Grundkenntnisse der OOP

1.2. Projekte

Ausgangspunkt für eine grafische Benutzeroberfläche ist eine von einem JFrame abgeleitete Klasse. In Eclipse kann mittels STRG-N der Assistent geöffnet werden. Dort kann unter WindowBuilder → Swing Designer → JFrame und der Eingabe eines geeigneten Klassennamens ein erstes Fenster erstellt werden.

Mit dem WindowBuilder Assistenten wird automatisch eine von der JFrame abgeleitete Klasse erstellt, die eine main()-Methode enthält die ein Objekt von der erstellte Klasse ableitet und dieses JFrame-Fenster sichtbar macht.

Gleichzeitig wird ein Konstruktor angelegt, der dem Fenster eine Standardgröße (und ein Layout – siehe später) zuweist.

Das Programm ist schon lauffähig und wenn es erstellt und ausgeführt wird, dann wird ein leeres Fenster mit einer Titelleiste und den Standard-Bedienelementen (Minimieren, Maximieren, Fenster-Schließen) erzeugt.

In der Entwicklungsoberfläche von Eclipse sind unter dem Code-Fenster zwei Reiter sichtbar: Source und Design. Mit dem WindowBuilder kann der generierte Code im Source-Fenster angezeigt werden und eine Darstellung des Fensters im Design-Fenster.

Das Design-Fenster gliedert sich wiederum in die Fenster-Ansicht und zusätzlich:

☑ Palette-Fenster: hier sind Komponenten zu finden, die via Drag-and-Drop in der Fenster-Ansicht plaziert werden können.

☑ Components-Fenster: hier ist eine hirarchische Darstellung der Komponenten auf der aktuellen Oberfläche zu finden. Gerade wenn das Projekt größer wird, dann hilft dieses Fenster sehr gut um Komponenten wieder zu finden.

☑ Properties-Fenster: dieser Bereich zeigt die Eigenschaften des aktuell in der Fenster-Ansicht gewählten Komponente an.

1.3. Komponenten

Komponenten die im Fenster platziert werden, werden sofort benannt. Es werden Abkürzungen verwendet, gefolgt von einem erklärenden Namen:

btn_OkButton für OK Schaltflächen (JButton)
btn_StartKonv für Schhaltflächen (JButton) die z.B. eine Konvertierung starten
lbl_Adr für ein Textfelder (JLabel) das eine Adresse enthält
...

1.3.1. JFrame, JPanel

Ein JFrame ist ein Fenster und wird automatisch vom WindowBuilder-Assistenten zu Beginn erstellt. Zum Erstellen von Komponenten in einem Fenster muss zuerst ein JPanel im JFrame erstellt werden. Der WindowBuilder macht das automatisch beim Erstellen einer JFrame-Klasse.

In den Eigenschaften eines JFrame gibt es unter Anderen folgenden Eigenschaften:

☑ title: Name des Fensters (Titelleiste)

☑ iconImage: Symbol des Fensters (links oben in der Titelleiste)

☑ resizeable: Fenster soll in der Größe veränderbar sein oder nicht.

☑ defaultCloseOperation: eintscheidet ob das Fenster sofort geschlossen werden soll oder nicht, wenn auf das x in der Titelleiste geklickt wird. Dadurch kann zum Beispiel beim Schließen noch die Abfrage eingebaut werden ob ein aktueller Stand gespeichert werden soll.

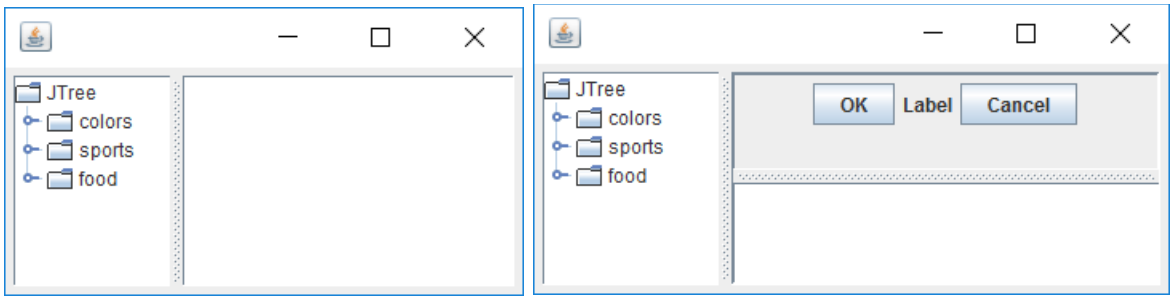
1.3.2. Kontainer/Containers

Kontainer sind Komponenten zur Organisation der Oberfläche.

☑ JSplitPane:ermöglicht eine Teilung eines Fensterbereichs mit Verschiebebalken.

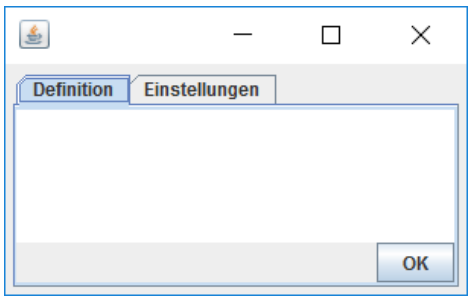
Linkes Beispiel: SplitPane – Verschiebebalken zwischen den beiden Komponenten (JTree und JTextArea).

Rechtes Beispiel: SplitPane mit zweiter SplitPane im rechten Bereich. Die orientation wurde hier auf VERTICAL_SPLIT gesetzt.



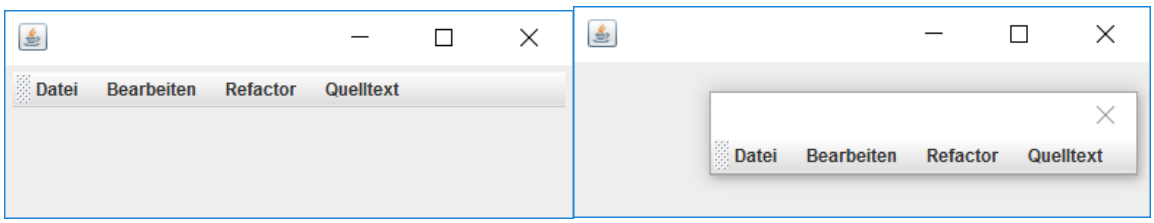
☑ JTabbedPane: Mehrere Bereiche übereinander, Auswahl durch Reiter.

Beispiel:



☑ JToolBar: Damit kann eine typische Menüleiste erstellt werden. Die Leiste kann automatisch an verschiedenen Stellen angedockt werden.

Beispiel:



In der rechten Abbildung ist die Toolbar abgedockt.

1.3.3. Layouts

Um Komponenten auf dem JPanel, JSplitBox oder JtabbedPane platzieren zu können, sollte zuerst ein geeignetes JLayout gewählt werden. Mit einem Layout kann die Lage von Komponenten bezogen zueinander eingestellt werden. Ganz entscheidend ist das Layout speziell für das Verhalten, wenn die Fenstergröße während der Laufzeit verändert wird. Sollen die Elemente zum Beispiel dadurch verschoben oder in ihrer Größe verändert werden.

Beispielhaft gibt es:

☑ Absolute Layout: Komponenten können beliebig auf dem Panel plaziert werden. Dieses Layout ist sehr gut geeignet um einfache Layouts zu erstellen.

☑ Border Layout: Es gibt 5 Bereiche in denen je ein Komponente plaziert werden kann. Innerhalb dieser Bereiche können sehr gut Container plaziert werden in denen mehrere Komponenten gesetzt werden. Auch in diesen Kontainern können/sollen Layouts gesetzt werden.

☑ GridBag Layout: Tabellarisches Layout. Eignet sich sehr gut zur Vertikalen und horizontalen Ausrichtung von Komponenten.

Obwohl ein Layout auch später verändert werden kann empfiehlt es sich das Layout zu Beginn zu setzen. Ein späteres Ändern im WindowBuilder kann ein Verlieren von erstellten Komponenten zur Folge haben. Allgemein bewährt sich ein häufiges Speichern oder Sichern des Layouts. Der WindowBuilder zerstört manchmal mit viel Aufwand erstellte Programme!

1.3.4. JLabel

Textkomponente zur Beschriftung. Labels können auch sehr gut für die Textausgabe und als farbige Blöcke verwendet werden.

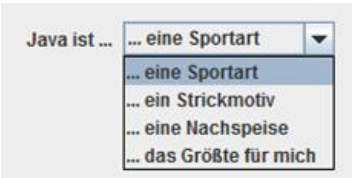
Methoden:
- setText("This and That"): verändert den Text eines Textfeldes
- getText(): ermittelt den Text eines Textfeldes

1.3.5. JButton

Schaltfläche. Die Beschriftung der Schaltfläche kann in der Eigenschaft text vorgegeben werden.

1.3.6. JComboBox

Auswahlelement. In der Eigenschaft model können die Werte eingetragen die vorgegeben werden sollen:



1.3.7. JCheckBox

1.3.8. JRadioButton

Grundsätzlich können Radio-Buttons genau wie Check-Boxen verwendet werden. Es ist jedoch üblich, dass mit Radio-Buttons nur Entscheidungen aus einer Gruppe ausgewählt werden können, die sich gegenseitig ausschließen. Dadurch werden bei Auswahl eines Elements automatisch die verbleibenden Radio-Butten-Elemente deaktiviert:

Jemand ist ein Robot oder man ist kein Robot, es geht weder Beides noch Keines. Um festzulegen welche Radio-Button-Elemente zusammengehören, müssen diese Elemente zusammen gruppiert werden (es könnten ja unterschiedliche Radio-Button-Elemente auf einer Seite zu finden sein). Im Structure Fenster können die zusammengehörenden Radio-Button-

Elemente gemeinsam markiert werden und im Kontextmenü mit Set Button Group → New standard verbunden werden.

1.3.9. JList

1.3.10. JTextField, JTextArea, JTextPane, JEditorPane

1.4. Events

Die Organisation der Komponente auf der Oberfläche ist der erste Schritt zur Erstellung einer GUI-Anwendung. Um der grafischen Darstellung eine Funktion hinterlegen zu können, ist eine Ereignisprogrammierung notwendig. Die Programmierung von grafischen Oberflächen unterscheidet sich von Konsolen-Anwendung dahingehend, dass eine grafische Oberfläche häufig solange unbeschäftigt bleibt, bis ein Benutzer eine Schaltfläche anklickt, einen Text in einer Textbox verändert, eine bestimmte Tastenkombination drückt oder der Mauszeiger über ein bestimmtes Feld bewegt wird. Eine solches Programm ist „ereignisgesteuert" programmiert. In Konsolenanwendungen gibt das Programm die Reihenfolge vor wann welcher Programmcode ausgeführt wird, in der ereignisgesteuerten Programmierung wird durch eintretende Ereignisse bestimmt was passiert.

Um Ereignissen (Englisch Event) eine bestimmte Logik zuzuordnen werden im Programm sogenannte Eventhandler hinterlegt die bestimmten Ereignissen zugeordnet werden.

Der WindowBuilder unterstützt auch hier:

> in der Fensteransicht wird für diejenige Komponente, für die ein Ereignis behandelt werden soll, im Kontextmenü (rechte Maustaste auf die Komponente) ausgewählt „Add event handler". Im erscheinenden Untermenü kann aus einer Vielzahl von möglichen Events ausgewählt werden.

Beispiel:

Für eine Schaltfläche (JButton) soll, wenn mit dem Mauszeiger darauf gezeigt wird, ein Text in der Konsole angezeigt werden:

☑ Rechte Maustaste auf die Schaltfläche → Add event handler → mouse → mouseEntered.

☑ WindowBuilder springt in die Source-Ansicht und erstellt für die ausgewählte Komponente einen addMouseListener und darin eine mouseEntered-Methode. Ein Listener ist ein Programmteil der warted (horcht) bis etwas passiert, in diesem Fall mit der Maus.

☑ Innerhalb der erstellten mouseEntered-Methode kann nun der Code eingefügt werden, der ausgeführt werden soll im Fall, dass das gewählte Ereignis (hier der Mauszeiger „betritt" die Schaltfläche. Für unser Beispiel würde man hier folgendes einfügen:

```
...
public void mouseEntered(MouseEvent arg0) {
    System.out.println("Maus auf Schaltfläche");
}
...
```

Häufig verwendete Events sind:

☑ mouseEntered: wird ausgelöst, wenn der Mauszeiger sich auf eine Komponente (JButton, JLabel, JTextField …) schiebt.

☑ mouseClicked: wird ausgelöst, wenn der Mauszeiger über der betroffenen Komponente befindet und die linke Maustaste gedrückt wird.

☑ actionPerformed: wird ausgelöst, wenn die Komponente (~gleichgültig wie) aktiviert wird.

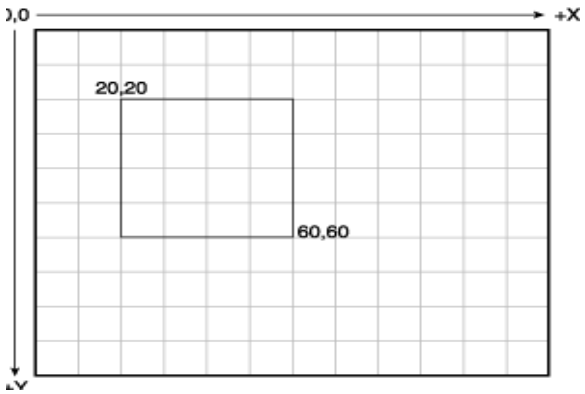
Aufgabe: Text Änderung

Frage:
Welches Event ist geeignet um für ein JTextField ein Ereignis auszulösen, wenn der Text verändert wurde?

Antwort:
keyTyped: wann immer eine Taste im Textfeld gedrückt wird, dann wird dieses Ereignis ausgelöst.
focusLost: wann immer ein aktiver Fokus vom betroffenen Textfeld abgeht (Tabulatortaste, Maus wählt andere Komponente …) dann wird dieses Ereignis ausgelöst.

Je nach Komponente kann es unterschiedliche Ereignisse geben.

1.4.1. Das grafische Koo



Hinweis: Titelleiste,... berücksichtigen

In den meisten Fällen steht dem Programm nicht das gesamte Fenster zur Ausgabe zur Verfügung, sondern es gibt Randelemente wie Titelzeilen, Menüs oder Rahmen, die nicht überschrieben werden können. Mit der Methode getInsets() kann die Größe dieser Randelemente ermittelt werden.

1.4.2. Die "Leinwand" und die Methode paint()

Um Linen, Rechtecke, … zeichnen zu können, brauchen Sie 2 Dinge:

1. Eine Leinwand
2. Funktionen/Methoden zum Zeichnen von Linien, …

Wir wollen uns zunächst um die Leinwand kümmern. Diese wird in java durch die Klasse Graphics repräsentiert.

☑ Die Methode getGraphics()

Man kann sich von beliebigen Dialog-Elementen (Jpanel, Jlabel, …) eine Leinwand erzeugen und auf dieser dann verschiedene Linien, Rechtecke,... zeichnen.

Im folg. Beispiel wird von einem JPanel der sog. Graphics-Kontext (die Leinwand) geholt und auf der Leinwand wird dann eine Linie gezeichnet.

```
...
java.awt.Graphics leinwand = jPanel.getGraphics();
leinwand.drawLine (0,0,100,100);
…
```

Aufgabe: Diagonale

Frage:
Gegeben ist ein JLabel mit dem Namen jLabelZeichnung. Zeigen Sie die notwendigen Anweisungen, um eine Linie von der linken oberen zur rechten unteren Ecke zu zeichnen.

Antwort:
??????????????

☑ **Die Methode void paint(Graphics g)**

Die Methode paint(graphics g) wird automatisch aufgerufen, wenn

1. das Fenster das erstemal gezeichnet wird und wenn
2. durch Benutzeraktionen ein Teil des Fensters wieder sichtbar wird, der bisher verdeckt war.
3. Das Fenster in seiner Größe verändert wird

Wenn Sie Grafikausgaben benutzen müssen Sie die Methode paint() (über)schreiben. Diese kann so aussehen

```
….
@Override
public void paint(Graphics g){
            //um eventuelle Dialogelemente (Buttons,...) zu zeichnen

            super.paint(g);

            // die eigentliche Grafik wird im jPanel angezeigt
            java.awt.Graphics leinwand= jPanel.getGraphics();
            leinwand.drawLine (0,0,100,100);
}
```

1.5. Grafikfunktionen

1.5.1. Ein Fenster: GrafikDemo.java

Wir wollen zunächst ein Window erzeugen, das aus folg. GUI-Komponenten besteht:

☑ jDesktopPane: Der Container, der die anderen GUI-Komponenten enthält

☑ jButtonStart: Um das Neu-Zeichnen dem Benutzer zu ermöglichen

☑ jPanel: Die "Leinwand" auf die gezeichnet werden soll.

Die Methode paint() ist die eigentliche Methode zum Zeichnen der Grafik.

Hier ein Beispiel:

```
/* GrafikDemo.java */

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

public class GrafikDemo extends JFrame {

      private static final long serialVersionUID = 1L;
      private JDesktopPane jDesktopPane = null;
      private JPanel jPanel = null;
      private JButton jButtonStart = null;

      /**
       * This method initializes jDesktopPane
       *
       * @return javax.swing.JDesktopPane
       */
      private JDesktopPane getJDesktopPane() {
            if (jDesktopPane == null) {
                  jDesktopPane = new JDesktopPane();
                  jDesktopPane.add(getJButtonStart(), null);
                  jDesktopPane.add(getJPanel(), null);
            }
            return jDesktopPane;
      }


      /**
       * This method initializes jButtonStart
       *
       * @return javax.swing.JButton
       */
      private JButton getJButtonStart() {
            if (jButtonStart == null) {
                  jButtonStart = new JButton();
                  jButtonStart.setBounds(new Rectangle(0, 0, 70, 20));
                  jButtonStart.setText("Start");
                  jButtonStart.addActionListener(new
                        java.awt.event.ActionListener() {
                            public void actionPerformed(
                                  java.awt.event.ActionEvent e) {

                                 // Window neu zeichnen lassen
                                 repaint();

                        }
                  });
            }
            return jButtonStart;
      }



      /**
       * This method initializes jPanel
       *
```

```
	 * @return javax.swing.JButton
	 */
	private JPanel getJPanel() {
		if (jPanel == null) {
			jPanel = new JPanel();
			jPanel.setBackground(Color.yellow);
			jPanel.setBounds(new Rectangle(0, 20, 640, 460));
		}
		return jPanel;
	}


	/**
	 * This method paints GrafikDemo the Window
	 *
	 * @return void
	 */
	public void paint(java.awt.Graphics g){
		//um die steuerelemente (Buttons,...) zu zeichnen

		super.paint(g);

		// die eigentliche Grafik wird im jPanel angezeigt
		java.awt.Graphics leinwand= jPanel.getGraphics();
		leinwand.drawLine (0,0,100,100);
	}


	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				GrafikDemo thisClass = new GrafikDemo();

	thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
				thisClass.setVisible(true);
			}
		});
	}


	/**
	 * This is the default constructor
	 */
	public GrafikDemo() {
		super();
		initialize();
	}

	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	private void initialize() {
```

```
            this.setSize(640, 480);
            this.setContentPane(getJDesktopPane());
            this.setBackground(Color.orange);
            this.setTitle("Java: Grafik Programmierung");
        }

}
```

Dieses Programm soll als Gerüst für die folgenden Programme dienen. Um die folg. Programme zu testen, braucht jeweils nur die paint() Methode ersetzt zu werden.

1.5.2. Linien

```
java.awt.Graphics;

public void drawLine(int x1, int y1, int x2, int y2)
```

Zieht eine Linie von der Position `(x1,y1)` zur Position `(x2,y2)`. Beide Punkte dürfen an beliebiger Stelle im Fenster liegen, das Einhalten einer bestimmten Reihenfolge ist nicht erforderlich. Teile der Ausgabe, die außerhalb des darstellbaren Bereichs liegen, werden, wie in grafikorientierten Systemen üblich, unterdrückt.

Das folgende Beispiel zeichnet eine Reihe von gleich hohen Linien, deren horizontaler Abstand durch einen Zufallszahlengenerator bestimmt wird. Das Ergebnis hat dadurch Ähnlichkeit mit einem Barcode (ist aber keiner):

```
/* Linien.inc */

public void paint(Graphics g)
{
  int i;
  int x = 80;

  for (i=0; i<60; ++i) {
    g.drawLine(x,40,x,100);
    x += 1+3*Math.random();
  }
}
```

1.5.3. Rechtecke

```
java.awt.Graphics;

public void drawRect(int x, int y, int width, int height)
```

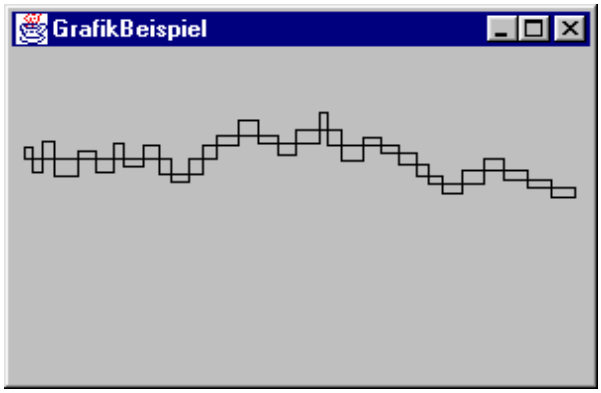
Zeichnet ein Rechteck der Breite `width` und der Höhe `height`, dessen linke obere Ecke an der Position `(x,y)` liegt. Eine größere Breite dehnt das Rechteck nach rechts aus, eine größere Höhe nach unten.

Das folgende Beispiel zeichnet eine Kette von nebeneinanderliegenden Rechtecken, deren Größe durch einen Zufallszahlengenerator bestimmt wird. Der Zufallszahlengenerator entscheidet auch, ob ein Rechteck an der Ober- oder Unterseite seines Vorgängers festgemacht wird:

```
/* Rechtecke.inc */

public void paint(Graphics g)
{
  int x = 10, y = 80;
  int sizex, sizey = 0;

  while (x < 280 && y < 180) {
    sizex = 4 + (int) (Math.random() * 9);
    if (Math.random() > 0.5) {
      y += sizey;
      sizey = 4 + (int) (Math.random() * 6);
    } else {
      sizey = 4 + (int) (Math.random() * 6);
      y -= sizey;
    }
    g.drawRect(x,y,sizex,sizey);
    x += sizex;
  }
}
```



1.5.4. Polygon

```
java.awt.Graphics;

public void drawPolygon(int[] arx, int[] ary, int cnt)
```

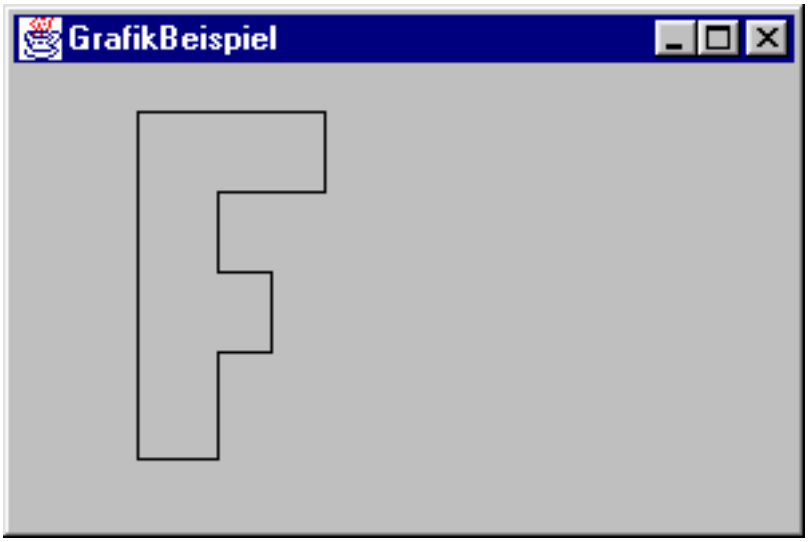
Mit Hilfe der Methode drawPolygon ist es möglich, Linienzüge zu zeichnen, bei denen das Ende eines Elements mit dem Anfang des jeweils nächsten verbunden ist:

drawPolygon erwartet drei Parameter. Der erste ist ein Array mit einer Liste der *x*-Koordinaten und der zweite ein Array mit einer Liste der *y*-Koordinaten. Beide Arrays müssen so synchronisiert sein, dass ein Paar von Werten an derselben Indexposition immer auch ein Koordinatenpaar ergibt. Die Anzahl der gültigen Koordinatenpaare wird durch den dritten Parameter festgelegt.

Das folgende Beispiel gibt den Buchstaben »F« mit Hilfe eines geschlossenen Polygons aus:

```
public void paint(Graphics g)
{
  int[] arx = {50,50,120,120,80,80,100,100,80,80};
  int[] ary = {170,40,40,70,70,100,100,130,130,170};

  g.drawPolygon(arx,ary,arx.length);
}
```



1.5.5. Kreis

```
java.awt.Graphics;

public void drawOval(int x, int y, int width, int height)
```

Die Graphics-Klasse von Java erlaubt sowohl das Zeichnen von Kreisen als auch von Ellipsen und Kreisabschnitten. Ein Kreis wird dabei als Spezialisierung einer Ellipse angesehen, und beide Objekte werden mit der Methode drawOval gezeichnet:

Anders als in anderen Grafiksystemen werden bei dieser Methode nicht der Mittelpunkt und der Radius des Kreises angegeben, sondern die übergebenen Parameter spezifizieren ein Rechteck der Größe `width` und `heigth`, dessen linke obere Ecke an der Position `(x,y)` liegt. Gezeichnet wird dann der größte Kreis, der vollständig in das Rechteck hineinpasst.

1.6. Farben

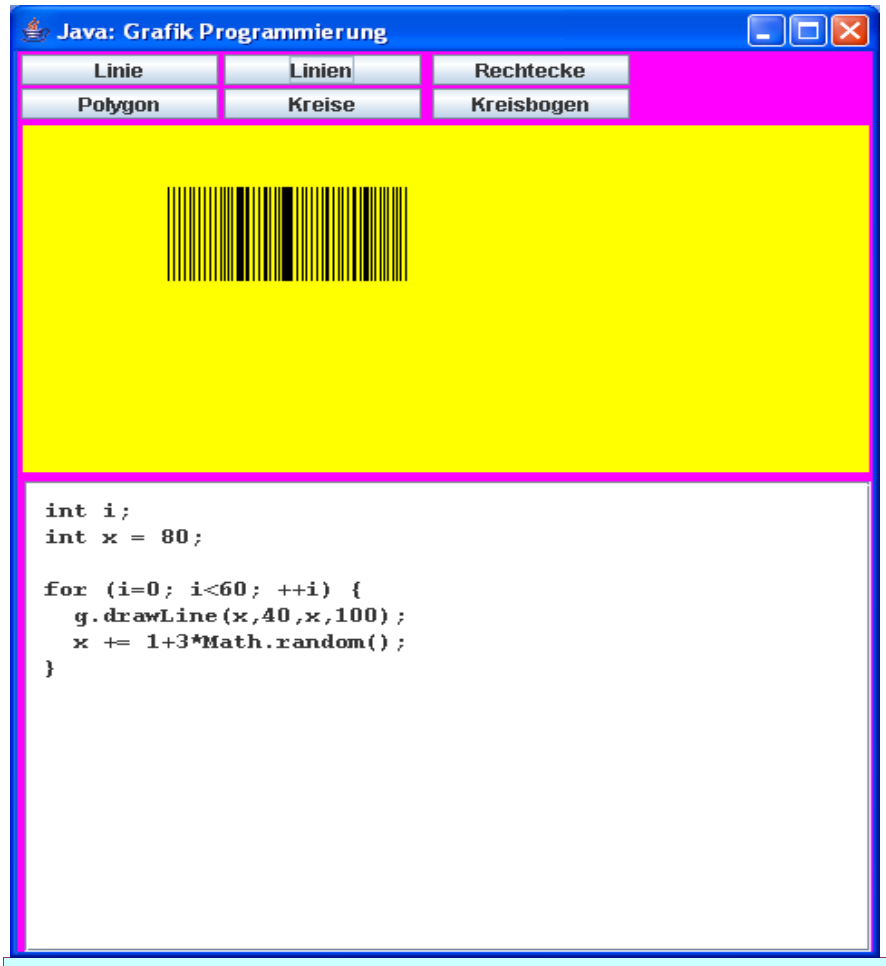
java.awt.Color

```
leinwand.setColor(java.awt.Color.yellow);
```

Weitere Informationen siehe www.javabuch.de

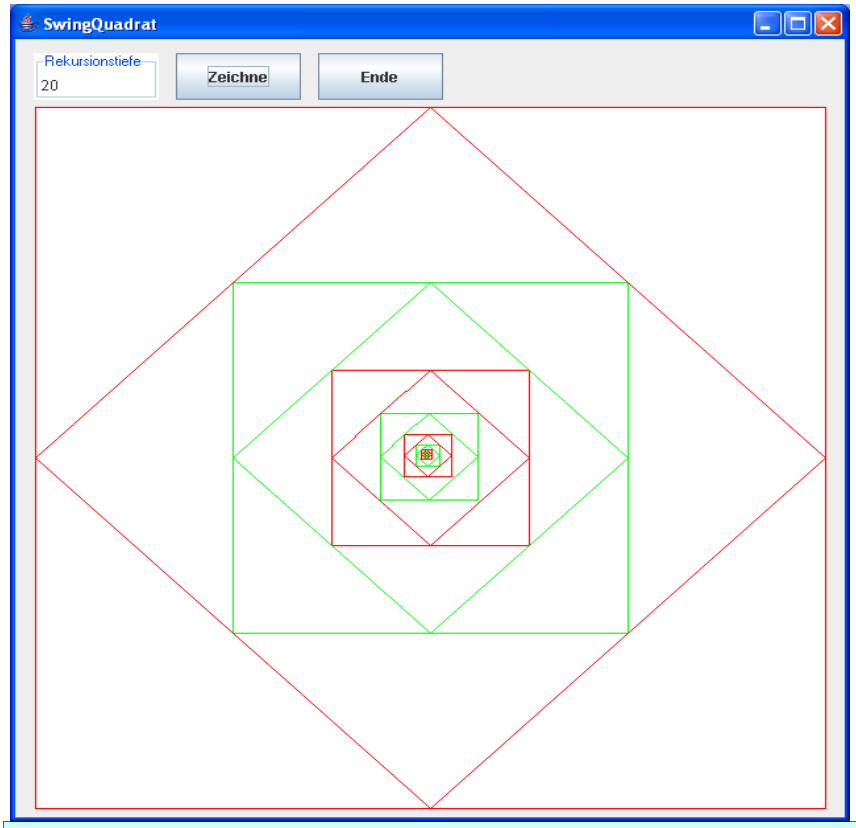
1.7. Erste Beispiele

1.7.1. +Aufgabe: DemoGrafik.java

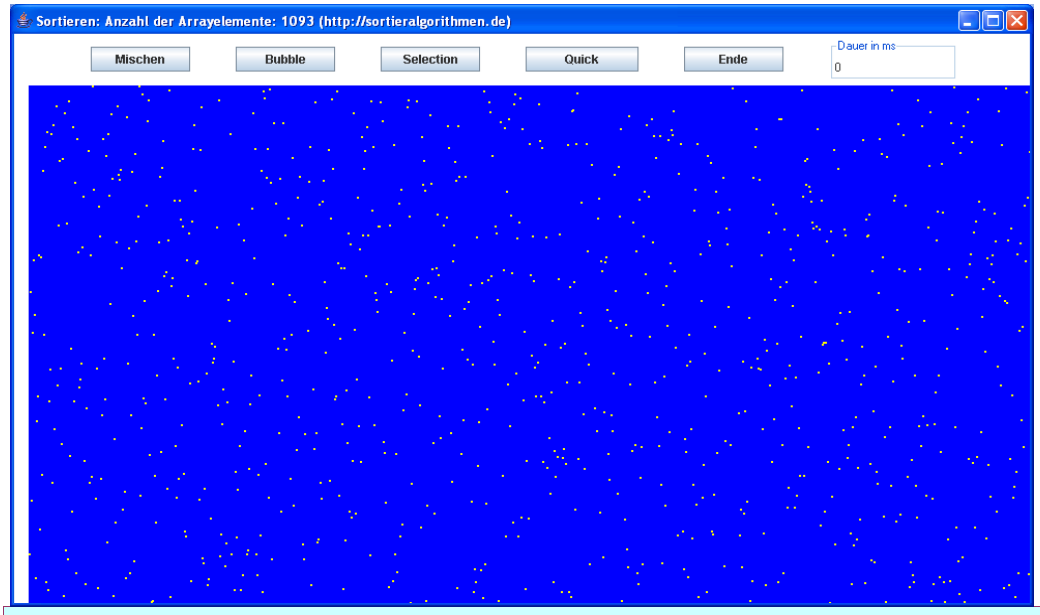


Linien (mit Diagonalen in versch. Farben) als Hausübung

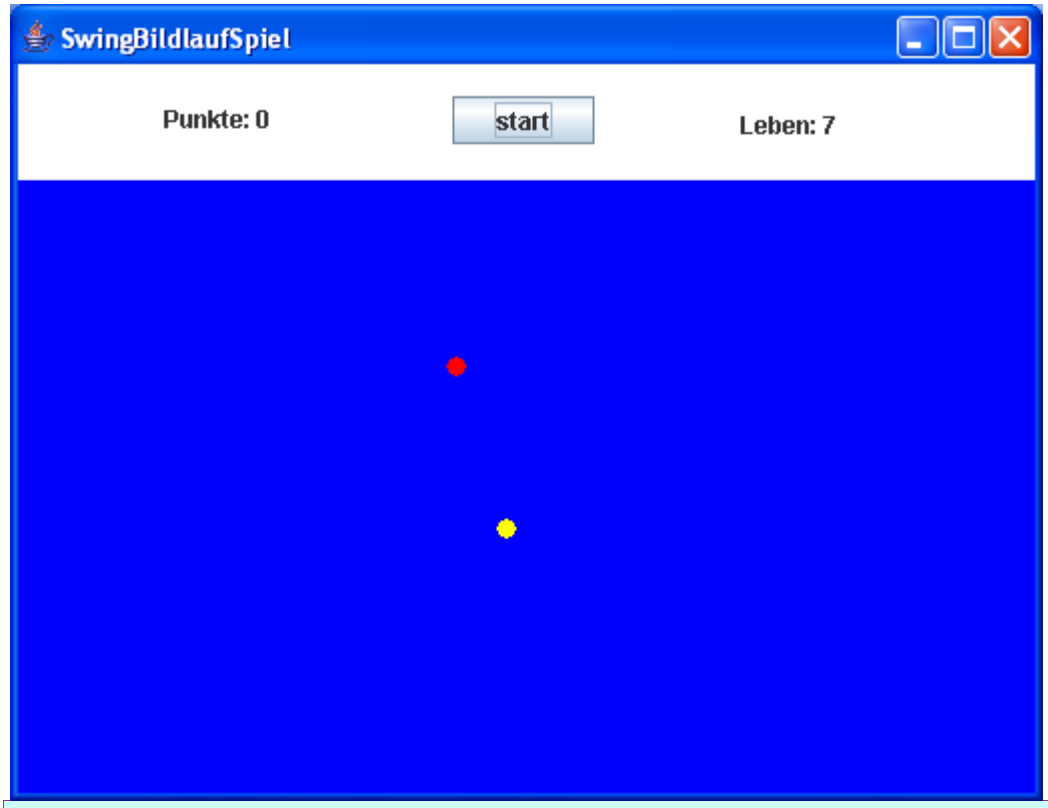
1.7.2. Aufgabe: SwingQuadrat.java



1.7.3. Aufgabe: VisualSort.java



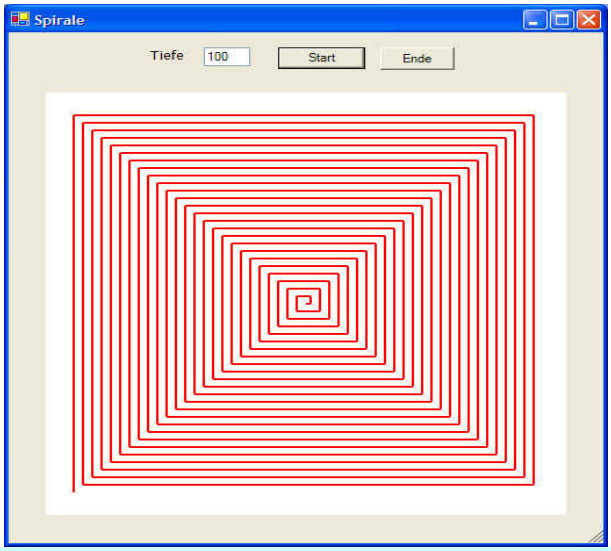
1.7.4. Aufgabe: SwingBildlaufSpiel.java



Start:
☑ 2 Bälle(Kreise) werden in die Mitte des Panels gesetzt.
☑ Es werden pro Ball unterschiedliche Richtungen per Zufallszahlen ermittelt.
☑ Angetrieben durch einen Timer bewegen sich die Bälle aus dem Panel.
☑ Der Spieler muss durch einen Mausklick auf den Ball dafür sorgen, dass der Ball wieder in der Panelmitte angeordnet wird.
☑ Sollte der Spieler den Ball nicht anklicken können und er verschwindet aus dem Panel verliert der Spieler ein Leben.

1.7.5. +Aufgabe: SwingSpirale

Wir wollen eine eckige Spirale zeichnen.



Unten sehen sie einen Auschnitt aus einem Qt-Programm, das diese Aufgabe erfüllt. Versuchen

Sie diese Aufgabe in VB.Net zu lösen.

```
void frmSpirale::btn_zeichne_clicked()
{
QPainter paint(this);
QPen p(red,1);
paint.setPen(p);

erase();

int ordnung, laenge ;
float winkel;
float x, y,x_old, y_old;

laenge=4;
winkel=0.0;
ordnung= atoi( cmb_anzahl->currentText());

x_old=320; y_old=240;

for (int i=1; i<= ordnung; i++) {
    x=x_old+laenge*cos(winkel);
    y=y_old-laenge*sin(winkel);

   paint.drawLine(round(x_old),round(y_old),round(x),round(y));
   x_old=x;
   y_old=y;
   laenge+=4;
   winkel+= M_PI / 2;
}
}
```

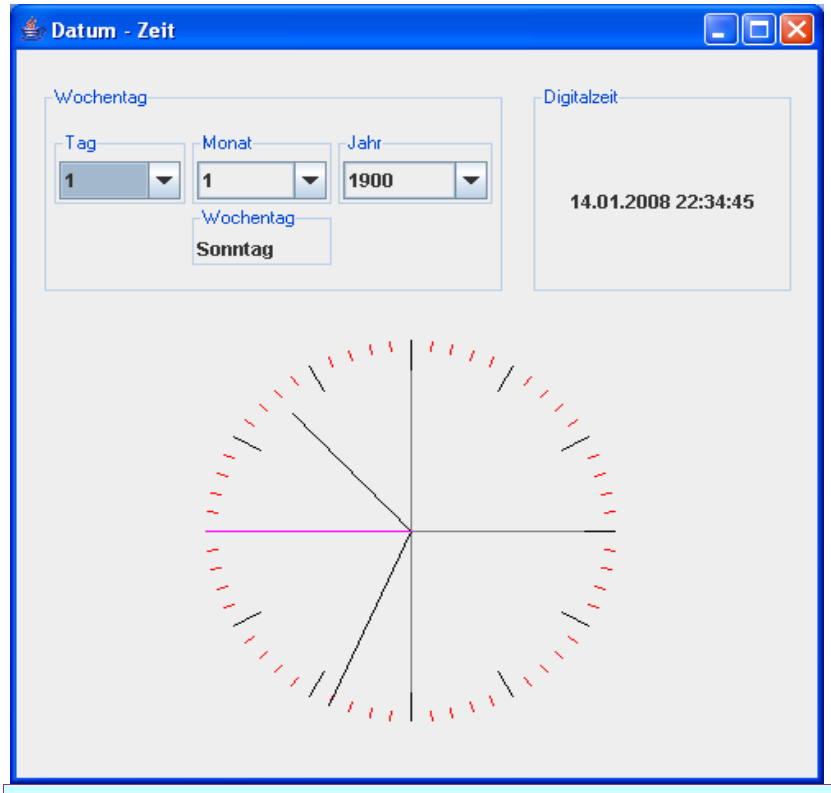
Frage:
Können Sie den Algorithmus beschreiben?

1.7.6. +Aufgabe: SwingLabyrinth

Wir wollen nach dem unten zu sehenden Beispiel ein Labyrinth (unter Verwendung von Zufallszahlen) zeichnen. Dieses werden wir später verwenden, um Algorithmen zu testen, die

einen möglichen Weg aus dem Labyrinth finden. (Backtracking). Es gibt links oben einen Eingang und rechts unten einen Ausgang.

1.7.7. +Aufgabe: SwingDatumZeit.java



Verschiedene Datumsfunktionen und Graphikfunktionen auf einen Blick.
Wochentagsberechnung, Aktuelle Uhrzeit.

1.7.8. Hinweis: Datum, Zeit

```
java.text.SimpleDateFormat df= new java.text.SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
jLabelDigitalZeit.setText(df.format(new java.util.Date()));
```

Hinweis: Formel zur Berechnung des Wochentages

w=[a+int((a/4) - int (a/100) + int(a/400) +d] mod 7;

a jahreszahl des vorjahres
d anzahl der tage im gefragten jahr (inkl. gewuenschter tag)
w= 0 so
w= 1 mo
w= 2 di
w= 3 mi
w= 4 do
w= 5 fr
w= 6 sa

Beispiel:
Auf welchen Wochentag fiel der 1.03.1984
a=1983
d= 31+29+1=61 (1984 ist ein Schaltjahr)

Hinweise zum Schaltjahr:
Jahr ist ein Schaltjahr, wenn

1. (Jahr durch 4 teilbar UND
2. Jahr nicht durch 100 teilbar)
3. ODER aber Jahr durch 400 teilbar

Beispiel:
1900 kein Schaltjahr aber 2000 ein Schaltjahr

1.7.9. +Hinweis: AnalogUhr

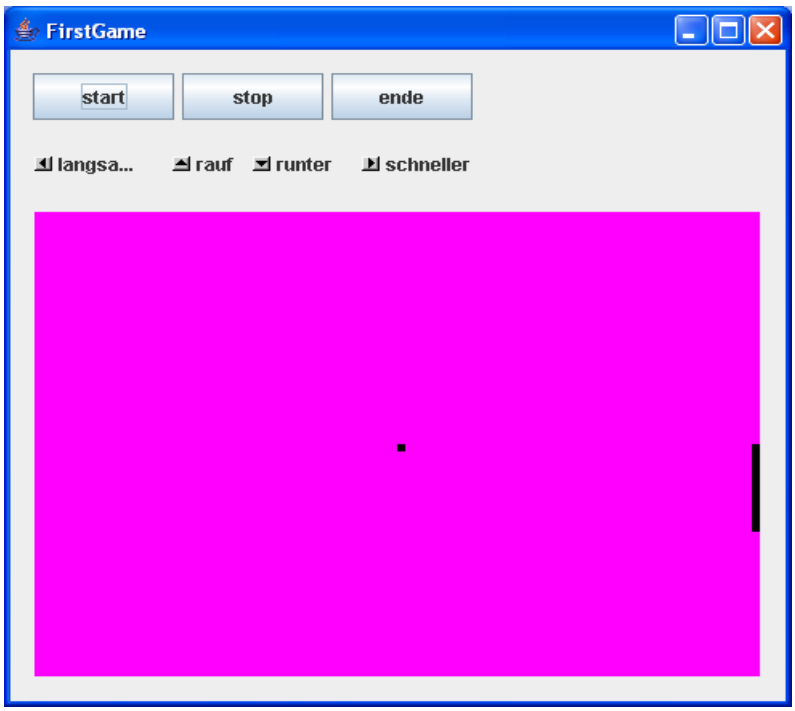
```
/* Mittelpunkt (h,k):
        x = h + r*cos(t)
        y = k + r*sin(t)
                */

// Stundenzeiger
grf.setColor (Color.black);
grf.drawLine(M.x, M.y,
                M.x + (int)((width/2 *0.85)*Math.cos(Math.toRadians(30*hrs - 90 + min/2))),
                M.y + (int)((height/2* 0.85)*Math.sin(Math.toRadians(30*hrs - 90 + min/2)))
                );

// Minutenzeiger
grf.setColor (Color.black);
grf.drawLine(M.x, M.y,
                        M.x + (int)((width/2)*Math.cos(Math.toRadians(6*min - 90))),
                        M.y + (int)((height/2)*Math.sin(Math.toRadians(6*min - 90)))
                        );
```

1.8. +Ein erstes Spiel: SpielFirstGame.java

Ein Ball soll sich in einem Spielfeld bewegen. Auf der linken,oberen und unteren Spielfeldseite wird der Ball zurück ins Spielfeld gespielt. Auf der rechten Seite ist ein kleines Brett, mit dem der Spieler den herankommenden Ball wieder ins Spielfeld schlagen soll. Verfehlt der Spieler den Ball, wandert der Ball auf der rechten Seite aus dem Spielfeld und das Spiel ist vorbei. Die gesamte Spieldauer wird ausgegeben. Der Spieler kann mit den Cursortasten UP,DOWN das Brett auf und ab bewegen und mit LEFT,RIGHT die Ballgeschwindigkeit verändern.



Am Spiel sind folgende Komponenten beteiligt:

☑ Spielfeld zeichnen: paint(), repaint()

☑ Timer

☑ Spieler macht eine Tastatur/Mauseingabe

1.8.1. Der Spieler

☑ Der Spieler macht eine Tastatur/Mauseingabe und ändert so z.B. die Richtung des Brettes - wird durch ein Variable festgelegt- durch die UP-Taste. Fertig.

1.8.2. Der Timer

☑ Der Timer wird nach einer bestimmten delay-Time aufgerufen. Er ruft eine Funktion auf, nennen wir sie z.B. actionGame(). Damit erhält der Programmierer die Gelegenheit einzugreifen. Er stellt z.B. fest, ob das Spiel zu Ende ist, ob ein Zähler zu erhöhen ist. Er setzt z.B. die Position des Balles fest, etc....

☑ Der Timer ruft dann repaint() auf, um das Spielfeld neu zu zeichnen. (Beschreibung s.u.)

☑ Zusammenfassung:

- Timer()
  actionGame()

  repaint()

1.8.3. public void paint(java.awt.Graphics g)

Um das Spielfeld, den Ball und das Brett zeichnen.

Zu beachten ist Folgendes:

☑ paint() wird vom System aufgerufen, wenn das Fenster neu zu zeichnen ist. (z.B. ein anderers Fenster hat "darübergelegen".

☑ Paint() wird vom Programmierer nicht direkt aufgerufen, sondern der Programmierer ruft repaint() auf, wenn das Fenster neu gezeichnet werden soll. (z.B. hat sich die Ballrichtung geändert). Repaint() kann natürlich auch durch einen Timer aufgerufen werden.

☑ repaint() wird vom Programmierer aufgerufen und ruft selbst update() auf. Wenn update() - eine von JFrame geerbte Methode- nicht überschrieben wird löscht diese das Fenster komplett und ruft selbst wieder paint() auf. Damit werden dann alle Komponenten und Container von JFrame neu gezeichnet und das Fenster "erstrahlt" mit all seinen Komponenten.

☑ Zusammenfassung:

- System: update() --> paint()
- Programmierer/Timer: repaint() --> update() --> paint()

1.8.4. Animation mit Timer und Bildschirmdarstellung

Bewegung/Animation kann gut mit einem Timer realisiert werden.

Timer:
actionGame()
repaint()

actionGame()
ändert zB. die Position der Spielfiguren,

Einen Timer definieren:

```
private javax.swing.Timer tm;
```

Timer-Eigenschaften (delay) setzen und Ereignisprozedur (actionPerformed()) zuordnen:

```
public void startTimer(){
tm= new javax.swing.Timer
      (
      delay,
      new java.awt.event.ActionListener()
      {
      public void actionPerformed (java.awt.event.ActionEvent e)
      {
            actionGame();

            repaint();
      }
      }
      );

tm.start();
}
```

1.8.5. Tastatureingaben, Cursortasten

Werden vom Spieler gemacht. Hier werden meist kleine Änderungen gemacht. Z.B. die Richtung ändern (move=-1 bzw. +). Die tatsächliche Anzeige wird in

Timer()

actionGame()

repaint()

gemacht.

```
....
jButtonStart.addKeyListener(new java.awt.event.KeyAdapter() {
      public void keyPressed(java.awt.event.KeyEvent e) {
            System.out.println("keyPressed()");
                  switch (e.getKeyCode()) {
                        case java.awt.event.KeyEvent.VK_UP:
                              // Spielbrett nach oben bewegen
                              move=-1;
                              break;
                        case java.awt.event.KeyEvent.VK_DOWN:
                              // Spielbrett nach unten bewegen
                              move=1;
                             break;

                        case java.awt.event.KeyEvent.VK_RIGHT:
                              // Timer soll schneller schalten
                              if (delay>=5)delay -=5;
                                    tm.setDelay(delay);
                              break;

                        case java.awt.event.KeyEvent.VK_LEFT:
                              // Timer soll langsamer schalten
                              delay +=5;
                              tm.setDelay(delay);
                              break;

                  }
            }
      });
```

Für unser Spiel brauchen wir noch folgende Teile:

Figur (Ball):

Die Position (auch Location genannt) wird durch die X,Y-Position angegeben.

private int ballX,ballY;

Die Grösse (auch Size genannt) wird z.B. durch die Breite, Höhe angegeben.

private int ballBreite,ballHoehe;

Richtung/Bewegung:

Die Richtung, in die sich die Figur bewegt wird durch die Schrittweite in X-/Y-Richtung angegeben.

```
private int ballSchrittweiteX,ballSchrittweiteY;
```

Nur horizontale Bewegung:

```
ballSchrittweiteY=0;
```

Steil nach rechts-unten:

```
ballSchrittweitex +=1;
ballSchrittweiteY +=2;
```

Die neue Position bestimmen:

```
altX= ball.getLocation().X;
altY= ball.getLocation().Y;
```

Die Ballrichtung ändern:

```
ballSchrittweiteX *=-1;
ball.setLocation(altX+  ballSchrittweiteX, altY +  ballSchrittweiteY);
```

Es hat sich was am Billdschirmaufbau geändert, drum zeichne neu.

```
repaint();
```

1.9. +Bildschirm Zwischenspeicherung

Oft erscheint ein Flackern auf dem Bildschirm. Dies liegt daran, dass die geerbte Methode update() das Fenster löscht und dann alle Inhalte neu zeichnet.

```
repaint()
        update() -- löscht
                paint()  -- neu zeichnen
```

Besser wäre es, wenn nur die Dinge gezeichnet würden, die sich tatsächlich verändert haben. Diese Methode muss allerdings für jede Problemstellung neu realisiert/programmiert werden. Deshalb wollen wir in der Folge die Doppelpufferung besprechen:

Mit folgender Methode kann man das Flackern verhindern. (Quelle: http://javacooperation.gmxhome.de/BildschirmflackernDeu.html)

```
repaint()
        update() -- löscht nicht, sondern
                speichert den Bildschirminhalt als Bild im RAM (OffScreen, Zwischenspeicher),
                ändert Dinge im Zwischenspeicher und zeichnet  Zwischenspeicher neu.
```

Die Doppelpufferung

"Beim Doppelpuffern wird bei jedem Animationsschritt zunächst die gesamte Bildschirmausgabe in ein Offscreen-Image geschrieben.
Erst wenn alle Ausgabeoperationen abgeschlossen sind, wird dieses Offscreen-Image auf die Fensteroberfläche kopiert. Im Detail sind dazu folgende Schritte erforderlich:

1. Das Fensterobjekt beschafft sich durch Aufruf von **createImage()** ein Offscreen-Image und speichert es in einer Instanzvariablen (= Erzeugen eines leeren Bildes)

2. Durch Aufruf von **getGraphics()** wird ein Grafikkontext zu diesem Image beschafft
3. Alle Bildschirmausgaben (inklusive Löschen des Bildschirms) gehen zunächst auf den Offscreen-Grafikkontext (= **Zeichnen des Bildes im Hintergrund**)
4. Wenn alle Ausgabeoperationen abgeschlossen sind, wird das Offscreen-Image mit **drawImage()** in das Ausgabefenster kopiert. (= Zeichnen des Bildes im Vordergrund)

Durch diese Vorgehensweise wird erreicht, dass das Bild komplett aufgebaut ist, bevor es angezeigt wird. Da beim anschließenden Kopieren die neuen Pixel direkt über die alten kopiert werden, erscheinen dem Betrachter nur die Teile des Bildes verändert, die auch tatsächlich geändert wurden. Ein Flackern, das entsteht, weil Flächen für einen kurzen Zeitraum gelöscht und dann wieder gefüllt werden, kann nicht mehr auftreten.

Einziger, gravierender Nachteil der Doppelpufferung ist jedoch, dass die Konstruktion des OffscreenImages ziemlich speicheraufwendig ist, und außerdem die Bilddaten zweimal geschrieben werden. In den meisten Fällen und auf schnellen Rechnern ist es jedoch weitaus vorteilhafter, in sehr kurzer Zeit die Doppelpufferung zu realisieren, als sich lange mit der Suche nach alternativen Möglichkeiten aufzuhalten.

So, aber nach der ganzen Theorie nun zur Implementierung der Doppelpufferung.

Der Programcode zur Umsetzung der Doppelpufferung

```
// Definition zweier Instanzvariablen für die Doppelpufferung im Kopf
// des Programmes

private Image dbImage;     // DoubleBuffer - Offscreen
private Graphics dbg;      // Grafikkontext des Offscreens
```

... anderer Programcode ...

Update - Methode, Realisierung der Doppelpufferung zur Reduzierung des Bildschirmflackerns

```
public void update (Graphics g)
{
      // Initialisierung des DoubleBuffers
      if (dbImage == null)
      {
            dbImage = createImage ( this.getSize().width,
                                    this.getSize().height);
            dbg = dbImage.getGraphics ();
      }

       // AB HIER wird mit dem Hintergrundspeicher gearbeitet
       //
      // Bildschirm im Hintergrund löschen
      dbg.setColor (getBackground ());
      dbg.fillRect (0, 0, this.getSize().width, this.getSize().height);
```

```
        // Auf gelöschten Hintergrund Vordergrund zeichnen
        dbg.setColor (getForeground());

        paint (dbg);

        // Nun fertig gezeichnetes Bild Offscreen auf dem richtigen
        // Bildschirm anzeigen
        g.drawImage (dbImage, 0, 0, this);

}
```

1.9.1. +Aufgabe: SpielFirstGame

Beschreibung s.oben

1.9.2. +Aufgabe: SpielSnake

1.9.3. +Aufgabe: SpielTetris

1.9.4. +Aufgabe: SpielPacman

1.10. Zusammenfassung

In diesem Kapitel wurden folgende Themen behandelt:

- Das Abstract Windowing Toolkit und das Paket java.awt
- Grundlagen der Grafikausgabe und Anlegen eines Fensters
- Die Methode paint und die Klasse Graphics
- Das grafische Koordinatensystem von Java

- Die Methoden drawLine, drawRect, drawPolygon, drawPolyline, drawOval und drawArc der Klasse Graphics zur Ausgabe von Linien, Rechtecken, Polygonen, Kreisen und Kreisbögen
- Die Bedeutung von Linien- und Füllmodus
- Die Methoden fillRect, fillRoundRect, fillPolygon, fillOval und fillArc zum Erzeugen von gefüllten Flächen
- Kopieren und Löschen von Flächen mit den Methoden clearRect und copyArea
- Die Clipping-Region und die Methoden clipRect, setClip, getClip und getBounds

1.11. Ausblick

Strings/Streams/Files

Aufgabenverzeichnis

Aufgabe: Diagonale 3
Frage: 3
Gegeben ist ein JLabel mit dem Namen jLabelZeichnung. Zeigen Sie die notwendigen Anweisungen, um eine Linie von der linken oberen zur rechten unteren Ecke zu zeichnen... 3
Antwort: 3
????????????? 3
+Aufgabe: DemoGrafik.java 10
Aufgabe: SwingQuadrat.java 11
Aufgabe: VisualSort.java 11
Aufgabe: SwingBildlaufSpiel.java 12
Start: 12
2 Bälle(Kreise) werden in die Mitte des Panels gesetzt. 12
Es werden pro Ball unterschiedliche Richtungen per Zufallszahlen ermittelt. 12
Angetrieben durch einen Timer bewegen sich die Bälle aus dem Panel. 12
Der Spieler muss durch einen Mausklick auf den Ball dafür sorgen, dass der Ball wieder in der Panelmitte angeordnet wird. 12
Sollte der Spieler den Ball nicht anklicken können und er verschwindet aus dem Panel verliert der Spieler ein Leben. 12
+Aufgabe: SwingSpirale 12
Wir wollen eine eckige Spirale zeichnen. 12
Unten sehen sie einen Auschnitt aus einem Qt-Programm, das diese Aufgabe erfüllt. Versuchen Sie diese Aufgabe in VB.Net zu lösen. 13

```
void frmSpirale::btn_zeichne_clicked()
{
QPainter paint(this);
QPen p(red,1);
paint.setPen(p);

erase();

int ordnung, laenge ;
float winkel;
float x, y,x_old, y_old;

laenge=4;
winkel=0.0;
ordnung= atoi( cmb_anzahl->currentText());

x_old=320; y_old=240;
```

```
for (int i=1; i<= ordnung; i++) {
   x=x_old+laenge*cos(winkel);
   y=y_old-laenge*sin(winkel);

  paint.drawLine(round(x_old),round(y_old),round(x),round(y));
  x_old=x;
  y_old=y;
  laenge+=4;
  winkel+= M_PI / 2;
}
} .....13
```

Frage:
Können Sie den Algorithmus beschreiben? 14
+Aufgabe: SwingLabyrinth 14
Wir wollen nach dem unten zu sehenden Beispiel ein Labyrinth (unter Verwendung von Zufallszahlen) zeichnen. Dieses werden wir später verwenden, um Algorithmen zu testen, die einen möglichen Weg aus dem Labyrinth finden. (Backtracking). Es gibt links oben einen Eingang und rechts unten einen Ausgang. 14
+Aufgabe: SwingDatumZeit.java 14
+Aufgabe: SpielFirstGame 21
Beschreibung s.oben 21
+Aufgabe: SpielSnake 22
+Aufgabe: SpielTetris 22
+Aufgabe: SpielPacman 22