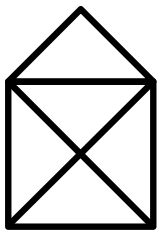
Grafik-Programmierung

In dieser Übung beschäftigen wir uns zunächst mit elementaren Grundlagen der Grafikprogrammierung. In der nächsten Übung werden wir dies auf Spiele anwenden.

Aufgabe 1: Einfache Grafik: Haus vom Nikolaus



Zeichnen Sie das Haus vom Nikolaus.

a) Erstellen Sie dazu eine von **`JPanel`** abgeleitete Klasse ***`HausPanel`*** im Paket **`exercises.gui.haus`**.

- Überschreiben Sie dort die Methode
  `public void` ***`paintComponent`*****`(Graphics g)`**
  und zeichnen Sie in das übergebene Graphics-Objekt `g` das Haus.

  Hinweis: Zum Zeichnen von Linien können Sie die Methode
  `drawLine(x1, y1, x2, y2)`
  aus der Klasse **`Graphics`** verwenden.

b) Erzeugen Sie ein **`JFrame`**-Fenster, fügen Sie ein Objekt der Klasse **`HausPanel`** zu diesem JFrame hinzu und zeigen Sie das Fenster auf dem Bildschirm an.

`exercises.gui.haus.Haus.java:`

```
package exercises.gui.haus;

import java.awt.Color;

import javax.swing.JFrame;

public class Haus {

	public static void main(String[] args) {
		JFrame frame = new JFrame();

		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		HausPanel panel = new HausPanel();
		panel.setBackground(Color.white);
```

```
            frame.add(panel);

            frame.setSize(500, 500);
            frame.setVisible(true);
        }
    }
```

exercises.gui.haus.HausPanel.java:

```
    package exercises.gui.haus;

    import javax.swing.JPanel;

    import java.awt.BasicStroke;
    import java.awt.Color;
    import java.awt.Graphics;
    import java.awt.Graphics2D;

    public class HausPanel extends JPanel {

        @Override
        public void paintComponent(Graphics g) {
            Graphics2D g2d = (Graphics2D) g;
            g2d.setStroke( new BasicStroke(3.5f) );

            g.setColor(Color.blue);
            g.drawRect(100, 200, 100, 100);
            g.drawLine(100, 200, 150, 150);
            g.drawLine(150, 150, 200, 200);
            g.drawLine(100, 200, 200, 300);
            g.drawLine(200, 200, 100, 300);

        }
    }
```

Aufgabe 2: Einfache Grafik: Olympische Ringe

Zeichnen Sie eine die Olympischen Ringe (in vereinfachter Form):

a) Erstellen Sie dazu eine von **`JPanel`** abgeleitete Klasse ***`RingPanel`*** im Paket **`exercises.gui.ring1`**.

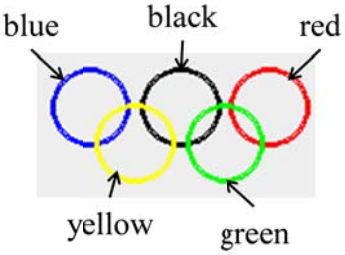
- Übergeben Sie der Klasse **`RingPanel`** die Startposition (des ersten Kreises) und Größe der Kreise im Konstruktor und speichern Sie sich diese Werte in Attributen.

- Überschreiben Sie dann die Methode
  `public void paintComponent(Graphics g)`
  und zeichnen Sie in das übergebene Graphics-Objekt `g` die Ringe mit der spezifizierten Größe an die vorgegebene Position.

Hinweise:

- Zum Zeichnen von Kreisen können Sie die Methode
  `drawOval(x, y, width, height)`
  aus der Klasse **`java.awt.Graphics`** verwenden.

- Die Farben der einzelnen Ringe sind jeweils wie folgt definiert. (Passende Farbwerte finden Sie als Konstanten in der Klasse **`java.awt.Color`**.)



- Zum Festlegen der Strichdicke können Sie die Methode
  `setStroke(new BasicStroke( <<Strichdicke>> )`
  aus der Klasse **`Graphics2D`** verwenden.

b) Erzeugen Sie (in einer Klasse **`exercises.gui.ring1.Rings`**) ein **`JFrame`**-Fenster, fügen Sie ein Objekt der Klasse **`RingPanel`** zu diesem JFrame hinzu und zeigen Sie das Fenster auf dem Bildschirm an.

`exercises.gui.ring1.Rings.java:`

```
package exercises.gui.ring1;
```

```
import javax.swing.JFrame;

public class Rings {

	public static void main(String[] args) {
		JFrame myframe = new JFrame();

	   myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		RingPanel panel = new RingPanel();

		myframe.add( panel );

		myframe.setSize(400, 400);
		myframe.setVisible(true);
	}
}
```

exercises.gui.ring1.Rings.java:

```
package exercises.gui.ring1;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JPanel;

public class RingPanel extends JPanel {

	private static final long serialVersionUID = 2L;

	@Override
	public void paintComponent(Graphics g) {
		((Graphics2D) g).setStroke( new BasicStroke(3.0f) );

		g.setColor(Color.blue);
		g.drawOval(100, 100, 50, 50);

		g.setColor(Color.black);
		g.drawOval(160, 100, 50, 50);

		g.setColor(Color.red);
		g.drawOval(220, 100, 50, 50);

		g.setColor(Color.yellow);
		g.drawOval(130, 130, 50, 50);

		g.setColor(Color.green);
		g.drawOval(190, 130, 50, 50);
	}
}
```

Aufgabe 3: Sich selbst zeichnende Ring-Objekte

Ändern Sie nun Ihr Programm dahingehend ab, dass die einzelnen Ringe nicht mehr direkt in der Klasse **`RingPanel`** gezeichnet werden. Erstellen Sie stattdessen eine Klasse **`SingleRing`**, welche jeweils einen einzelnen Ring repräsentiert und sich selbst zeichnen kann.

a) Erstellen Sie die Klasse **`exercises.gui.ring2.SingleRing`** zur Repräsentation eines einzelnen Ringes. Als Attribute enthält sie die Farbe des Ringes, dessen Größe sowie dessen Position (x und y-Koordinate).

   Damit Sie die modifizierte Version von der Ausgangsversion ihrer Klassen unterscheiden können, packen Sie Ihre modifizierten Klassen dieses Aufgabenteils in das Paket **`exercises.gui.ring2.`**

   An Methoden soll die Klasse SingleRing bereitstellen:

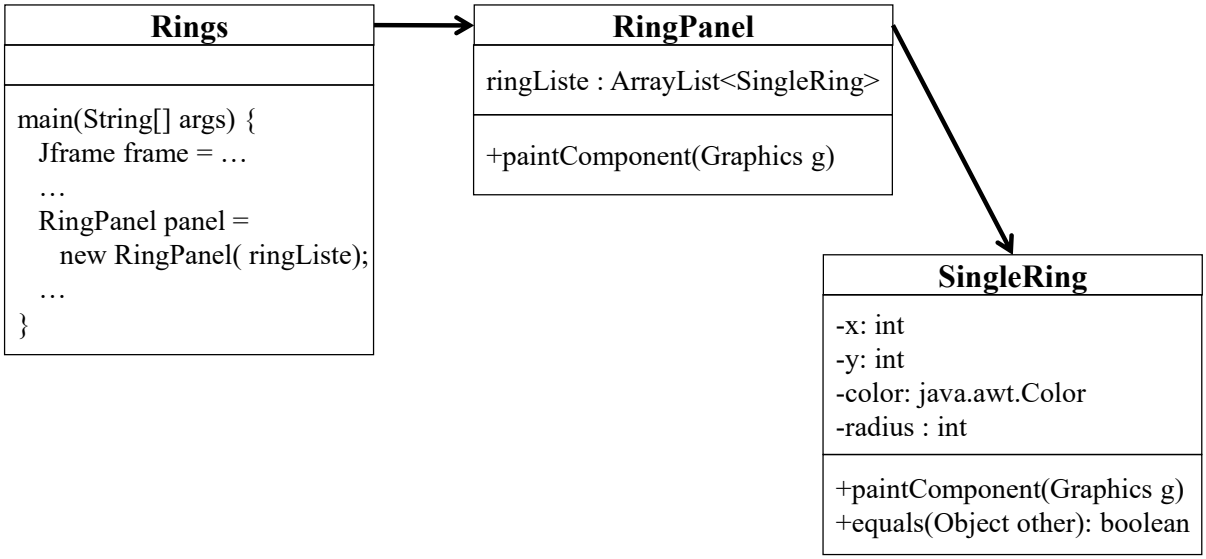
   - **`public void paintComponent(Graphics g)`** zeichnet sich selbst
   - **`public boolean equals(Object other)`** testet auf Gleichheit

b) Modifizieren Sie die Klasse **`RingPanel`** derart, dass dort nicht mehr die Größe und Anfangsposition der Ringe gespeichert wird, sondern stattdessen eine Liste von **`SingleRing`**-Objekten. Zeichnen Sie alle Objekte dieser Liste in der Methode **`paintComponent`**.

   Hinweis: Für Listen bietet Java u.a. die generische Klasse **`ArrayList`** (**`java.util.ArrayList<SingleRing>`**, siehe http://docs.oracle.com/javase/7/docs/api/index.html?java/util/ArrayList.html). Diese besitzt u.a. die folgenden Funktionen:
   - **`new ArrayList<SingleRing>()`** erzeugt eine leere Liste von SingleRing-Objekten.
   - **`boolean add(SingleRing elem)`** fügt ein Element zur Liste hinzu
   - **`for (SingleRing elem : liste) { … }`** iteriert über alle Elemente einer Liste von SingleRing-Elementen.
   - **`SingleRing get(int index)`** liest das Element an Position **`index`** aus
   - **`int size()`** liefert die Anzahl der Elemente in der Liste
   - **`boolean remove(SingleRing elem)`** löscht das Element aus der Liste, sofern es enthalten ist.

c) Erzeugen Sie in der Klasse **`exercises.gui.ring2.Rings`** die fünf verschiedenen **`SingleRing`**-Objekte und übergeben Sie diese (als

`ArrayList<SingleRing>`) an ihr modifiziertes **`exercises.gui.ring2.RingPanel`**-Objekt.



`exercises.gui.ring2.Rings.java:`

```
package exercises.gui.ring2;

import java.awt.Color;
import javax.swing.JFrame;
import java.util.LinkedList;

public class Rings {

      public static void main(String[] args) {
            JFrame myframe = new JFrame();
            myframe.setDefaultCloseOperation(
                              JFrame.EXIT_ON_CLOSE);

            SingleRing ring1 =
                        new SingleRing(100, 100, 50, 5, Color.blue);
            SingleRing ring2 =
                        new SingleRing(160, 100, 50, 5, Color.black);
            SingleRing ring3 =
                        new SingleRing(220, 100, 50, 5, Color.red);
            SingleRing ring4 =
                        new SingleRing(130, 130, 50, 5, Color.yellow);
            SingleRing ring5 =
                        new SingleRing(190, 130, 50, 5, Color.green);

            LinkedList<SingleRing> liste =
                        new LinkedList<SingleRing>();
            liste.add( ring1 );
            liste.add( ring2 );
            liste.add( ring3 );
            liste.add( ring4 );
```

```
                liste.add( ring5 );

                liste.remove(1);  // entferne ring2
                liste.add( new SingleRing(160, 100, 50, 5, Color.black));
                                  // fehlenden Ring wieder hinzufügen

                RingPanel panel = new RingPanel( liste );

                myframe.setSize(400, 400);
                myframe.setVisible(true);
          }
     }
```

exercises.gui.ring2.RingPanel.java:

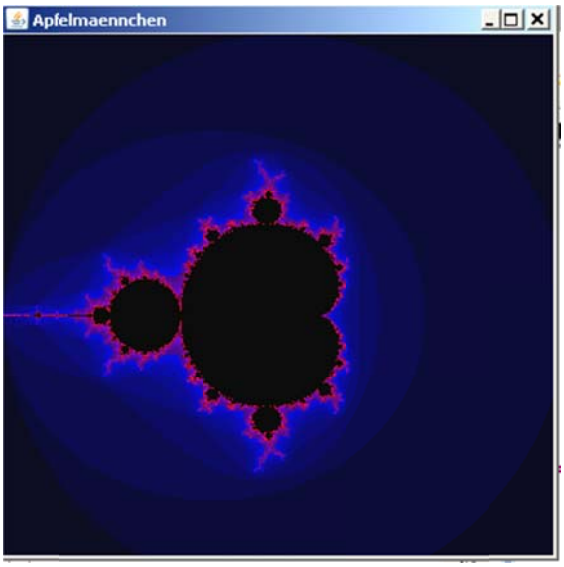
```
     package exercises.gui.ring2;

     import java.awt.Graphics;
     import java.util.LinkedList;

     import javax.swing.JPanel;

     public class RingPanel extends JPanel {

          private static final long serialVersionUID = 2L;

          LinkedList<SingleRing> liste;

          RingPanel(LinkedList<SingleRing> liste) {
                super();
                this.liste = liste;
          }

          @Override
          public void paintComponent(Graphics g) {
                for (SingleRing ring : liste) {
                      ring.paintComponent(g);
                }
          }
     }
```

exercises.gui.ring2.SingleRing.java:

```
     package exercises.gui.ring2;

     import java.awt.BasicStroke;
     import java.awt.Graphics;
     import java.awt.Graphics2D;
     import java.awt.Color;

     public class SingleRing {

          private int x;
```

```
        private int y;
        private int radius;
        private int randbreite;
        private Color color;

        public SingleRing(int x, int y,
                    int radius, int randbreite,
                    Color color) {
              this.x = x;
              this.y = y;
              this.radius = radius;
              this.randbreite = randbreite;
              this.color = color;
        }

        public void paintComponent(Graphics g) {
              g.setColor(color);
              ((Graphics2D) g).setStroke(
                                    new BasicStroke(randbreite) );
              g.drawOval(x, y, radius, radius);
        }
  }
```

Zusatz-Aufgabe 4: Apfelmännchen



a) Schreiben Sie eine Klasse **`exercises.gui.apfel.Complex`**, welche komplexe Zahlen *c = x + iy* mit Realteil *x* und Imaginärteil *y* speichert und mindestens die folgenden Methoden bereitstellt:

- **`Complex(double x, double y)`** Konstruktor zum Setzen der komplexen Zahl
- **`Complex add(Complex other)`** zum Addieren einer Zahl
- **`Complex sqr()`** zum Quadrieren einer komplexen Zahl
- **`double abs()`** für den Absolutbetrag

Die Attribute sollen von außen nicht zugreifbar sein.

Hinweis: Für das Rechnen mit komplexen Zahlen $c_i = x_i + iy_i$ gilt:

$$c_1 + c_2 = (x_1 + iy_1) + (x_2 + iy_2) = (x_1 + x_2) + i(y_1 + y_2)$$

$$c^2 = (x + iy)^2 = (x^2 - y^2) + i(2 * x * y)$$

$$|c| = |x + y| = \sqrt{x^2 + y^2}$$

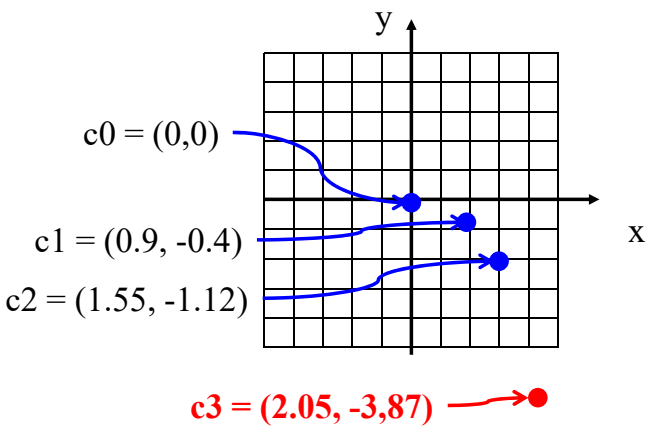
b) Komplexe Zahlen können als Punkte im kartesischen Koordinatensystem interpretiert werden, indem der Realteil der x-Koordinate und der Imaginärteil der y-Koordinate entspricht. Benutzen Sie nun die Klasse **`Complex`**, um ein **Apfelmännchen** grafisch darzustellen.

Dabei wird für eine (komplexe) Zahl *c* die Folge

$$c_0 = 0$$
$$c_{N+1} = c_N^2 + c$$

betrachtet. Diejenigen Zahlen *c*, bei denen die obige Folge beschränkt bleibt, d.h. deren Betrag der Folgenglieder nicht über alle Grenzen wächst, gehören zum Apfelmännchen (oder allgemeiner zur Mandelbrotmenge) und werden schwarz dargestellt. Bei den anderen Punkten wird der Farbwert in Abhängigkeit der Geschwindigkeit der Divergenz bestimmt.

Wir suchen deshalb des erste Glied c_N, welches betragsmäßig größer als 2 ist ($|c_N^2| > 2$). Sollte dies nach *N = 100* Schritten noch nicht der Fall sein, so brechen wir die Berechnung für diesen Punkt *c* ab und betrachten den Punkt als zum Inneren des Apfelmännchens zugehörig.



Die ermittelte Schrittzahl *N* wird verwendet, um die Farbe des Punktes *c* zu wählen.

Dabei wird die Farbe im RGB-System durch den roten, grünen und blauen Anteil jeweils als Zahl zwischen 0 und 255 angegeben. Dies kann mit der Klasse **`Color`** aus dem Paket **`java.awt`** erfolgen, welche einen geeigneten Konstruktor bereitstellt.

Zum Zeichnen des Apfelmänchens wählen wir *gruen = 0*, und für die anderen Farbanteile in Abhängigkeit von *N*:

$$color(c) = \begin{cases} rot = 0 & ,blau = N * 20 & ,falls\ 0 \leq N \leq 10 \\ rot = N * 5 & ,blau = N * 10 & ,falls\ 10 < N \leq 20 \\ rot = N * 4 & ,blau = N * 2 & ,falls\ 20 < N \leq 60 \\ rot = N * 3 & ,blau = 0 & ,falls\ 60 < N \leq 80 \\ rot = N * 2 & ,blau = 0 & ,falls\ 80 < N \leq 90 \\ rot = N * 2 & ,blau = (N - 90) * 25 & ,falls\ 90 < N < 100 \\ rot = 0 & ,blau = 0 & ,falls\ 100 \leq N \end{cases}$$

Zeichnen Sie das Apfelmänchen im Zahlenbereich $(-2, -2) \leq c \leq (2, 2)$ in einer Auflösung von 400x400 Punkten.

Beachten Sie, dass Sie die Koordinaten einer komplexen Zahl geeignet in Fensterpunkte umrechnen müssen. Außerdem benötigen Sie – wie in den vorangegangenen Aufgaben – ein Fenster (**`JFrame`**) und ein **`JPanel`** zum Zeichnen.

Apfel.java:

```
/*
 * Apfelmaennchen
 *
 * Thomas Nitsche, 29.6.01
 */

import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;


public class Apfel {
	public static void main(String[] args) {
		Apfelmann a = new Apfelmann();
		a.init();
	}
}
```

```
class Point {
	// lokale Koordinaten (als komplexe Zahl interpretiert)
	private double x;
	private double y;

	// maximale Ausdehnung, identisch fuer alle Punkte
	private static double xmin, xmax, ymin, ymax;
	private static double xrange, yrange;

	static void setRange(int xrange, int yrange) {
```

```
        Point.xrange = (double) xrange;
        Point.yrange = (double) yrange;
    }

    static void setMinMax(double xmin, double xmax,
                    double ymin, double ymax) {
        Point.xmin = xmin;
        Point.xmax = xmax;
        Point.ymin = ymin;
        Point.ymax = ymax;
    }

    static void defaultRange() {
        // Initialisierung Range
        setRange(400, 400);
        setMinMax(0, 400, 0, 400);
    }

    Point() {
        this.x = 0;
        this.y = 0;
    }

    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // addiere komplexe Zahl
    Point add(Point other) {
        Point result = new Point(this.x, this.y);
        result.x = result.x + other.x;
        result.y = result.y + other.y;
        return result;
    }

    // Quadrat der komplexen Zahl
    Point sqr() {
        double real = this.x * this.x - this.y * this.y;
        double imag = 2 * this.x * this.y;
        Point result = new Point(real, imag);
        return result;
    }

    // Absolutbetrag der komplexen Zahl
    double abs() {
        double result = Math.sqrt(this.x * this.x
                                  + this.y * this.y);
        return result;
    }


    void paint(Color color, Graphics g, JPanel imagePanel) {
        // Skaliere aktuelle Koordinaten
        int xint, yint;
```

```
            xint = (int)( (x - xmin) / (xmax - xmin) * xrange);
            yint = (int)( (ymin - y) / (ymin - ymax) * yrange);

//          pad.setColor(color);
//          pad.drawDot(xint, yint);
            g.setColor(color);
            g.drawLine(xint, yint, xint, yint);
            //imagePanel.repaint();
      }

      public String toString() {
            return "(" + x + ", " + y + ")";
      }
}
```

```
class Apfelmann extends JFrame {

      int checkPoint(Point p) {
            int loops = 0;
            Point temp = new Point();

            do {
                  // temp = (temp.sqr()).add(p);
                  loops++;
                  temp = temp.sqr();
                  temp = temp.add(p);
            } while ((temp.abs() <= 2) && (loops < 100));

            return loops;
      }

      void paintPoint(double x, double y, Graphics g,
                      JPanel imagePanel)
      {
            Point p = new Point(x, y);
            int loops = checkPoint(p);
            Color col;

            if (loops == 100)    { col =
                           new Color(0, 0, 0);
                           // Color.black;
                           }
            else if (loops > 90) { col =
                           new Color(loops*2, 0, (loops-90)*25); }
            else if (loops > 80) { col = new Color(loops*2, 0, 0); }
            else if (loops > 60) { col = new Color(loops*3, 0, 0); }
            else if (loops > 20) { col = new Color(loops*4, 0,
                                                   loops*2); }
            else if (loops > 10) { col = new Color(loops*5, 0,
                                                   loops*10); }
            else           { col = new Color(0, 0, loops*20); }
```

```
            p.paint(col, g, imagePanel);
      }


      final int RANGE = 400;

      void init() {

//          Pad pad;

            int Ixrange, Iyrange, Ixmin, Ixmax, Iymin, Iymax;

            double xrange, yrange, xmin, xmax, ymin, ymax;
            // double xrange, yrange;
            double xstep, ystep;


//          pad = new Pad();

            // initialisieren
            Ixrange = RANGE;
            Iyrange = RANGE;
            Ixmin = 0;
            Ixmax = Ixrange;
            Iymin = 0;
            Iymax = Iyrange;

////        pad.setPadSize(Ixrange, Iyrange);
////        pad.setTitle("Apfelmaennchen");
////        pad.setVisible(true);
//          Pad.setHeight(Iyrange);
//          Pad.setWidth(Ixrange);
//          Pad.initialize("Apfelmaennchen");
//          Pad.setVisible(true);
            this.setSize(Ixrange, Iyrange);
            this.setTitle("Apfelmaennchen");
            this.setLocation(200, 100);
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setContentPane(getJContentPane());

            show();
            requestFocus();
            toFront();

            this.setVisible(true);

            xmin = -2.0 + (4.0 * Ixmin) / Ixrange;
            xmax = -2.0 + (4.0 * Ixmax) / Ixrange;
            ymin =  2.0 - (4.0 * Iymin) / Iyrange;
            ymax =  2.0 - (4.0 * Iymax) / Iyrange;
            xstep = (xmax - xmin) / Ixrange;
            ystep = (ymax - ymin) / Iyrange;

            Point.setRange(Ixrange, Iyrange);
            Point.setMinMax(xmin, xmax, ymin, ymax);
```

```
//          pad.clear();


            // Schleife
            //Graphics g = imagePanel.getGraphics();
            Graphics g = jContentPane.getGraphics();

            for (double x = xmin; x < xmax; x = x + xstep) {
                  for (double y = ymin; y > ymax; y = y + ystep) {
                        paintPoint(x, y,  g, imagePanel);
                  }
            }
      }

      private JPanel jContentPane = null;
      private JPanel imagePanel = null;

      private JPanel getImagePanel() {
            if (imagePanel == null) {
                  imagePanel = new JPanel();

                  imagePanel.setBackground(Color.white);
                  imagePanel.setForeground(Color.black);

                  imagePanel.setSize(RANGE, RANGE);
            }
            return imagePanel;
      }

      private JPanel getJContentPane() {
            if (jContentPane == null) {
                  jContentPane = new JPanel();

                  jContentPane.setBackground(Color.white);
                  jContentPane.setForeground(Color.black);


//                jContentPane.setLayout(new BorderLayout());
                  jContentPane.setLayout(null);
//                jContentPane.setSize(RANGE, RANGE);
                  jContentPane.setBounds(0, 0, RANGE, RANGE);

                  // imagePanel
//                jContentPane.add(getImagePanel(),
                                    BorderLayout.NORTH);
                  jContentPane.add(getImagePanel());
                  imagePanel.setBounds(0, 0, RANGE, RANGE);

            }
            return jContentPane;
      }

}
```

Zusatz-Aufgabe 5 (für Java-Experten): Java-Rätsel - Exception

Was geben die folgenden Java-Programm aus – und warum?

a) Unentschieden:

```
public class Unentschieden {
	public static void main(String[] args) {
		System.out.println(decision());
	}

	static boolean decision() {
		try {
			return true;
		} finally {
			return false;
		}
	}
}
```

Ausgabe:

```
false
```

b) ExceptionTest:

```
public class ExceptionTest {
	private ExceptionTest internalInstance =
					new ExceptionTest();

	public ExceptionTest() throws Exception {
		throw new Exception("Ich kehre nicht zurück");
	}

	public static void main(String[] args) {
		try {
			ExceptionTest o = new ExceptionTest();
			System.out.println("Überraschung!");
		} catch (Exception ex) {
			System.out.println("Ich habe es ja gesagt");
		}
	}
}
```

Ausgabe:

```
Exception in thread "main" java.lang.StackOverflowError
	at ExceptionTest.<init>(ExceptionTest.java:3)
	...
	at ExceptionTest.<init>(ExceptionTest.java:3)
```