Java

Fraktale

Alfred Nussbaumer

Der streng objektorientierte Ansatz von Java lässt ein direktes Umsetzen von Pascal-Programmtexten meistens nicht zu. Im Unterricht beliebte Beispiele zur **rekursiven Darstellung von Fraktalen** müssen daher neu formuliert werden. In diesem Beitrag soll eine Möglichkeit aufgezeigt werden, wie ein Graphik-Objekt korrekt durch Rekursionen hindurch weitergereicht wird. Darüber hinaus sollen Rekursionen und die Dimension von fraktalen Gebilden behandelt werden.

1. Turtle-Grafik

Fraktale Geometrie wird durch Algorithmen beschrieben, die beispielsweise in einer Art „Rückkopplung“ definiert sind. Um dies realisieren zu können, setzen wir das Verständnis von **Rekursionen** und die Verwendung einer einfachen **Turtlegrafik** voraus.

Von einer **Rekursion** sprechen wir, wenn eine Methode sich selber aufruft. Dieser Vorgang muss mit einer bestimmten Abbruchbedingung beendet werden. Verringert man bei jedem Aufruf etwa die Größe eines Objekts, so erhält man einen wiederholten Aufruf einer Zeichenvorschrift und die Möglichkeit, die Rekursion abzubrechen, wenn eine bestimmte Größe des Objekts unterschritten wird.

Von frühen Programmiersprachen her (vgl. LOGO) wird eine einfache „**Turtlegrafik**“ überliefert: Eine „Schildkröte“ bewegt sich über das Zeichenfeld und hinterlässt dabei die gewünschte Zeichenspur... Wir verwenden davon nur zwei Zeichenoperationen, die wir (in Anlehnung an die Turtlegrafik von LOGO) `fd()` und `rt()` nennen: Mit `fd(strecke)` bewegt sich die Turtle ein gerades Stück der Länge strecke weiter, `rt(winkel)` dreht die Schildkröte um den angegebenen Winkel:

Turtle.java

```
import java.awt.*;

class Turtle {
  double x;
  double y;
  double alpha;
  Container c;
  Graphics g;

  Turtle(Container ct, double x, double y) {
    c = ct;
    g = c.getGraphics();
    g.setColor(Color.black);
    this.x = x;
    this.y = y;
    alpha = 0;
  }

  public void fd (double strecke) {
    double aa  = alpha * Math.PI / 180;
    double dx = strecke * Math.cos(aa);
    double dy = strecke * Math.sin(aa);
    g.drawLine((int) x, (int) y, (int) (x+dx), (int) (y+dy));
    x = x+dx;
    y = y+dy;
  }

  public void rt (double winkel) {
    alpha = alpha - winkel;
  }
}
```

In der Klasse `Turtle` verwenden wir die Instanzvariablen `x` und `y`: Damit wird die Startposition der Turtle festgelegt. Die Klasse `Container` stellt ein Objekt zur Verfügung, das andere Objekte enthalten kann; hier ist das die Zeichenfläche `g`. Mit der Methode `getGraphics()` wird dieses Zeichenobjekt an eine Instanz der Klasse `Graphics` übergeben.

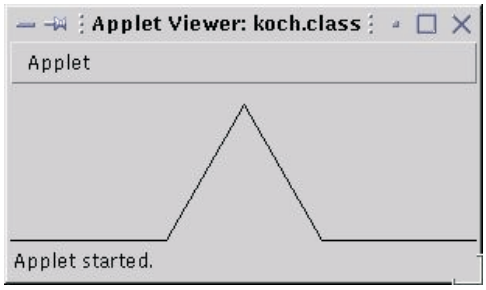
In der Methode `fd(strecke)` wird zunächst der „aktuelle“ Winkel der Turtle ins Bogenmaß umgerechnet. Aus der Strecke und dem Winkel wird mit Hilfe der Winkelfunktionen der Zuwachs `dx` und `dy` in den beiden Koordinatenrichtungen bestimmt und schließlich die Linie von der alten Position `(x,y)` zur neuen Position `(x+dx, y+dy)` gezeichnet. Die neue Position wird anschließend in den Variablen `x` und `y` gespeichert.

In der Methode `rt(winkel)` wird der negative Drehwinkel einfach zum aktuellen Winkel addiert (das Vorzeichen des Drehwinkels ergibt sich aus der Orientierung der y-Achse von „oben nach unten“).

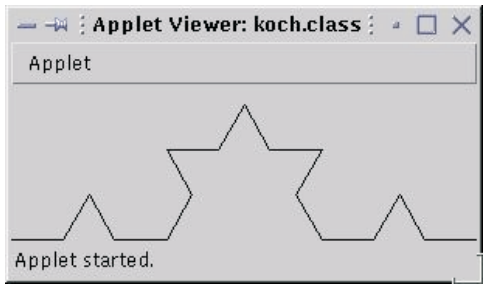
Damit die Klasse `Turtle` von anderen Java-Programmen verwendet werden kann, wird sie mit einem `import`-Befehl in den Programmcode eingebettet. Im einfachsten Fall steht dabei die Klasse `Turtle` im gleichen Verzeichnis wie das aufrufende Programm.

2. Die Kochkurve (`koch.java`)

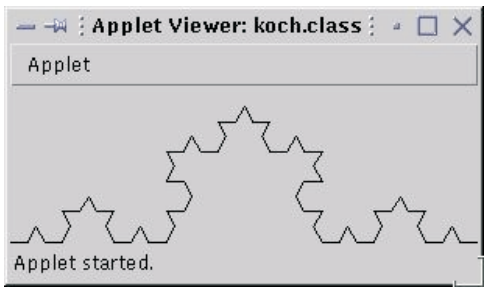
Ausgangspunkt der Kochkurve ist eine gerade Linie, die in drei gleiche Strecken geteilt wird. Über dem mittleren Teilstück wird ein gleichseitiges Dreieck errichtet.



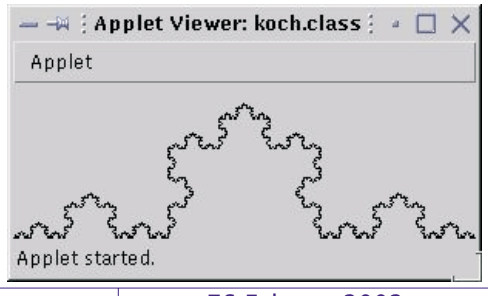
Durch Rekursion erhält man die gleiche Figur längs jeder Verbindungsstrecke.



Auf jedes gerade Teilstück der nun entstandenen Kurve lässt sich das Verfahren (rekursiv) wiederholen - es entsteht eine immer stärker „gezackte“ Kurve.



Rufen wir das Teilen der geraden Linie in drei Teile und das Zeichnen der vier (!) neuen Strecken dreimal auf, so sprechen wir auch von der **Rekursionstiefe 3**. Damit der Prozess abbricht, muss bei jeder Rekursion überprüft werden, ob das Ende schon erreicht ist...



koch.java

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

public class koch extends Applet {

    public void paint (Graphics g) {
     Turtle t = new Turtle(this,0,100);
     kochkurve(t, 300, 5);
    }

    public void kochkurve (Turtle t,
          double strecke, int ebene) {
     if (ebene > 0) {
         kochkurve (t, strecke/3, ebene - 1);
         t.rt(60);
         kochkurve(t, strecke/3, ebene -1);
         t.rt(-120);
         kochkurve(t, strecke/3, ebene -1);
         t.rt(60);
         kochkurve(t, strecke/3, ebene -1);
     } else t.fd(strecke);
    }
}
```

(Die Klasse `Turtle` muss bei diesem Beispiel im gleichen Verzeichnis wie das Programm `koch.java` stehen)

Mit ähnlichen Algorithmen, die im Internet oder in der Literatur leicht nachgeschlagen werden können, lässt sich eine Reihe „prominenter" Linienfraktale erzeugen!

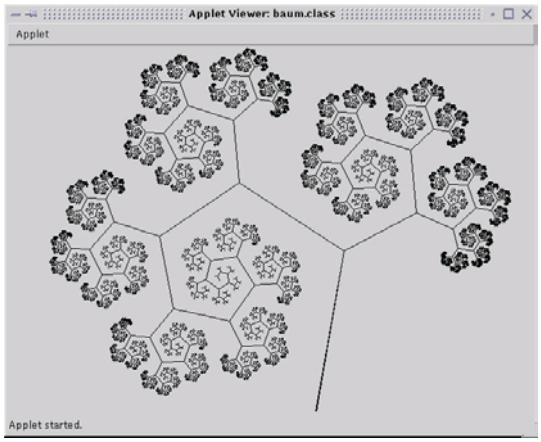
Bemerkung: Wo liegt der Unterschied?

```
public void spiral(Turtle t, double s, int w) {
  if (s < 200) {
    t.fd(s);
    t.rt(90);
    spiral(t, s + w, w);
  }
}

public void spirol(Turtle t, double s, int w) {
  if (s < 200) {
    spirol(t, s + w, w);
    t.fd(s);
    t.rt(90);
  }
}
```

3. Baumfraktale

Ein fraktaler Baum: (`baum.java`)

Zunächst wird eine Strecke nach oben gezeichnet (daher wird vor dem ersten Aufruf der rekursiven Methode der Winkel um 80° erhöht). Nach einer Drehung nach rechts um einen bestimmten Winkel wird die Strecke verkürzt weitergezeichnet. Dann kehrt man um, dreht zweimal den Winkel nach links, zeichnet die verkürzte Strecke, kehrt exakt um und kehrt zum Verzweigungspunkt zurück. Durch den rekursiven Aufruf der Methode können viele Ebenen abgearbeitet werden - es entsteht ein stark verästeltes Fraktal:



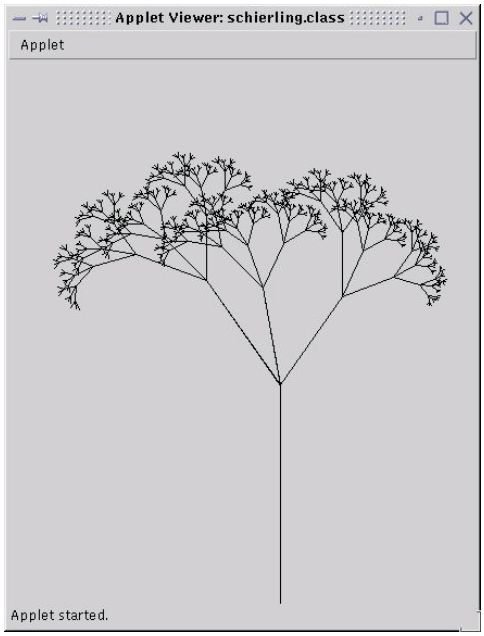

```
import java.awt.*;
import java.applet.*;
import Turtle.*;

public class baum extends Applet {
  public void paint (Graphics g) {
    Turtle t = new Turtle(this, 380,450);
    t.rt(80);
    baumz(t,200,52,15);
  }

  public void baumz(Turtle t, double stamm, double winkel, int ebene)
  {
    if (ebene > 0) {
     t.fd(stamm);
     t.rt(winkel+15);
     baumz(t, stamm/1.3, winkel, ebene - 1);
     t.rt(-2*winkel-15);
     baumz(t, stamm/2, winkel, ebene - 1);
     t.rt(winkel);
     t.fd(-stamm);
    }
  }
}
```

Aufgabe

Variiere den Anfangswinkel (im obigen Beispiel 80°), die Verkürzungsfaktoren (im Beispiel 1.3 bzw. 2) sowie die Winkel! Welche Rolle spielt die Angabe der Rekursionstiefe in der Variablen „`ebene`"?

Ein fraktaler Schierling (`schierling.java`)

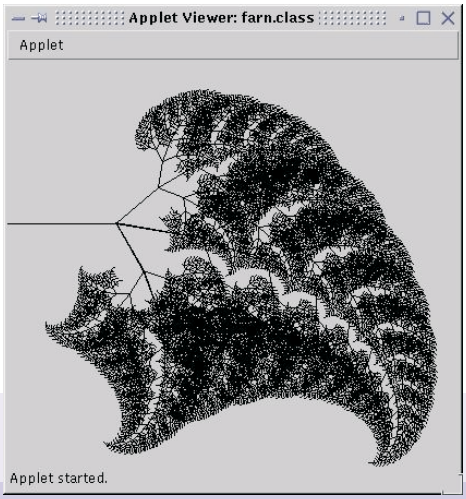
Dieses Fraktal ist ähnlich wie das „Baum-Fraktal" aufgebaut, allerdings treten 3 Verzweigungen auf, die zudem keinesfalls symmetrisch sind...




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

public class schierling extends Applet {
  public void paint (Graphics g) {
    Turtle t = new Turtle(this,250,500);
    t.rt(90);
    schierl(t,200,10,7);
  }

  public void schierl(Turtle d, double stamm,
                      double winkel, int ebene) {
    if (ebene > 0) {
       d.fd(stamm);
       d.rt(35);
       schierl(d,stamm/1.7, winkel, ebene - 1);
       d.rt(-70);
       schierl(d,stamm/2, winkel, ebene - 1);
       d.rt(35+winkel);
       schierl(d,stamm/2.2, winkel, ebene - 1);
       d.rt(-winkel);
       d.fd(-stamm);
    }
  }
}
```

Es fällt auf, dass die Fraktale ähnlich zu Strukturen sind, wie sie in der Natur, etwa in der Pflanzenwelt, auftreten.

Der fraktale Farn (farn.java)




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

public class farn
extends Applet {
  public void paint (Graphics g) {
    Turtle t = new Turtle(this,0,150);
    farnwedel(t,100);
  }

  public void farnwedel(Turtle t,double strecke)
  {
     if (strecke>1) {
         t.fd(strecke);
         t.rt(-60);
         farnwedel(t,strecke/2);
         t.rt(50);
         farnwedel(t,4*strecke/5);
         t.rt(50);
         farnwedel(t,strecke/2);
         t.rt(-40);
         t.fd(-strecke);
     }
  }
}
```

Beachte die verschiedenen „Verkürzungsfaktoren“ und die nicht symmetrischen Winkel! Überlege weiters genau die „Startbedingungen“ unter denen dieses Fraktal zu zeichnen begonnen wird! Welche Dimensionen muss der Rahmen für das Applet innerhalb der HTML-Datei bekommen, in der das Applet ausgegeben werden soll?

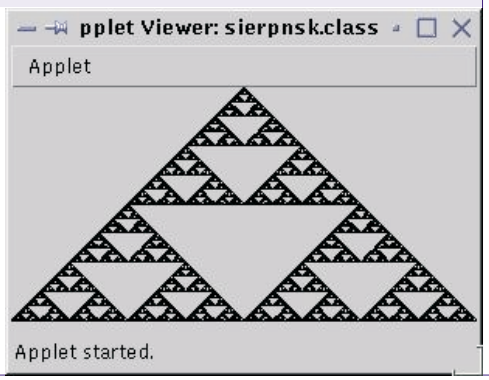
Bemerkung: Beim Zeichnen der Fraktale kann es vorkommen, dass die Grafikausgabe „ruckelt“ oder gar kurzzeitig zum Stillstand kommt. Die Ursache dafür liegt darin, dass der Java-Bytecode-Interpreter in bestimmten Abständen nicht mehr benötigten Hauptspeicher freigeben muss. Da bei tiefen Rekursionen jedoch (kurzfristig) zahlreiche Kopien der verwendeten Variablen im Hauptspeicher abgelegt werden müssen, fällt die Reorganisation des Hauptspeichers auf...

4. Fraktale ohne Turtle-Grafik

Beispiel: Sierpinsky-Dreieck (sierpnsk.java)

In diesem (und in den nächsten beiden Beispielen) verwenden wir nicht die Klasse **Turtle**. Die Rekursion kommt dadurch zu Stande, dass die Figur so lange verkleinert an einer neuen Position gezeichnet wird, bis die Basisgröße der Figur eine bestimmte Größe unterschritten hat...

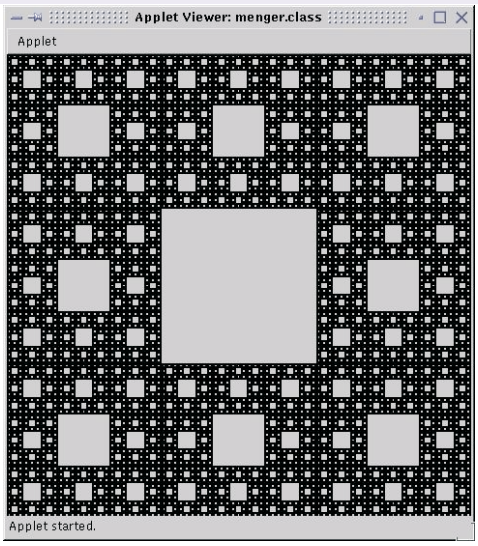
```
import java.awt.*;
import java.applet.*;
```




```
public class sierpnsk extends Applet {

  public void paint (Graphics g) {
    sierpinsky(this,300,150,0);
  }

  public void sierpinsky(Container ct, double c,
         double x, double y) {
    Graphics g = ct.getGraphics();
    if (c>2) {
     sierpinsky(ct, c/2, x-c/4, y+c/4);
     sierpinsky(ct, c/2, x+c/4, y+c/4);
     sierpinsky(ct, c/2, x, y);
     g.drawLine((int) x, (int) y, (int) (x-c/2), (int) (y+c/2));
     g.drawLine((int)(x-c/2),(int)(y+c/2),(int)(x+c/2),(int)(y+c/2));
     g.drawLine((int) (x+c/2), (int) (y+c/2), (int) x, (int) y);
    }
  }
}
```

Beispiel: Menger-Fraktal (menger.java)

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

public class menger extends Applet {

  public void paint (Graphics g) {
    mengers(this,240,240,240);
  }

  public void mengers(Container ct, double c,
         double x, double y) {
    Graphics g = ct.getGraphics();
    if (c > 2) {
         mengers(ct, c/3, x - 2*c/3, y - 2*c/3);
         mengers(ct, c/3, x, y - 2*c/3);
         mengers(ct, c/3, x + 2*c/3, y - 2*c/3);
         mengers(ct, c/3, x - 2*c/3, y);
         mengers(ct, c/3, x + 2*c/3, y);
         mengers(ct, c/3, x - 2*c/3, y + 2*c/3);
         mengers(ct, c/3, x, y + 2*c/3);
         mengers(ct, c/3, x + 2*c/3, y + 2*c/3);
    } else
         g.drawRect((int)(x-c), (int)(y-c),
              (int)(2*c), (int)(2*c));
  }
}
```



Bemerkung: Was ist die Dimension eines „fraktalen Gebildes“?

Wir betrachten zunächst die Dimension eines Quadrates mit der Seitenlänge l = 5. In dieses Quadrat fügen wir Quadrate mit der Seitenlänge l = 1 ein (der Verkürzungsfaktor beträgt also 5). Im gegebenen Quadrat haben 25 Quadrate Platz; daraus bestimmen wir die Dimension:

$$5^x = 25$$

Wir lösen diese Exponentialgleichung:

$$\lg(5^x) = \lg 25$$

$$x \lg 5 = \lg 25$$

$$x = \frac{\lg 25}{\lg 5} = 2$$

Somit ergibt sich für die Dimension der Wert 2 (das hätten wir auch gleich gewusst ;-).

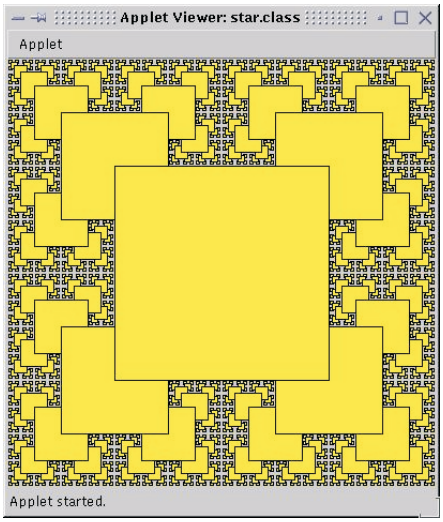
Wir wenden dieses Verfahren auf alle Fraktale an, etwa auf die Kochkurve: Man teilt die Strecke in 3 gleich lange Teilstrecken auf, wobei man anstatt des mittleren Teilabschnittes die beiden anderen Seiten des gleichseitigen Dreiecks zeichnet. Man erhält somit den Verkleinerungsfaktor 3, wobei 4 Strecken gezeichnet werden. Analog zur obigen Vorgangsweise erhalten wir:

$$x = \frac{\lg 4}{\lg 3} = 1{,}26186$$

Das Ergebnis ist keine ganze Zahl, sondern (näherungsweise) ein Bruch. Aus dieser Dimensionseigenschaft leitet sich der Name Fraktal ab!

(Menger: 1,89, Sierpinsky: 1,58 - rechne nach!)

Beispiel: star.java

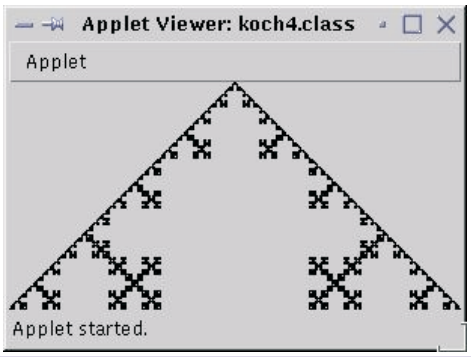



```
import java.awt.*;
import java.applet.*;

public class star extends Applet {

  public void paint (Graphics g) {
    starr(this,100,200,200);
  }

  public void starr(Container ct, double l,
     double x, double y) {
    Graphics g = ct.getGraphics();
    if (l>1) {
     starr(ct, l/2, x-l, y+l);
     starr(ct, l/2, x+l, y+l);
     starr(ct, l/2, x-l, y-l);
     starr(ct, l/2, x+l, y-l);
     g.setColor(Color.black);
     g.drawRect((int) (x-l),(int) (y-l),
          (int) (2*l), (int) (2*l));
     g.setColor(Color.red);
     g.fillRect((int) (x-l+1),(int) (y-l+1),
          (int) (2*l-1), (int) (2*l)-1);
    }
  }
}
```

Beachte, wie die fraktalen Strukturen mehrfärbig erstellt werden können (damit kann man beispielsweise sehr hübsche Baumfraktale, Farne oder Sierpinsky-Dreiecke zeichnen ;-)!

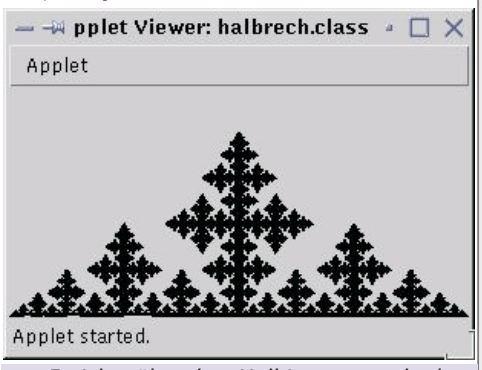
5. Weitere Aufgaben ;-)

Beispiel: „Kockkurve“ aus 5 Teilstücken



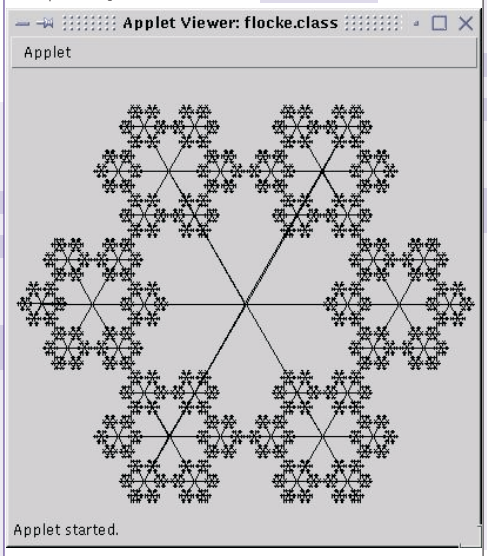
Teile eine Strecke in drei Teile und errichte über dem mittleren Teilstück ein Quadrat...

Beispiel



Errichte über dem Halbierungspunkt der Strecke eine Normale, die beispielsweise eine Länge von zwei Fünftel der ursprünglichen Strecke hat...

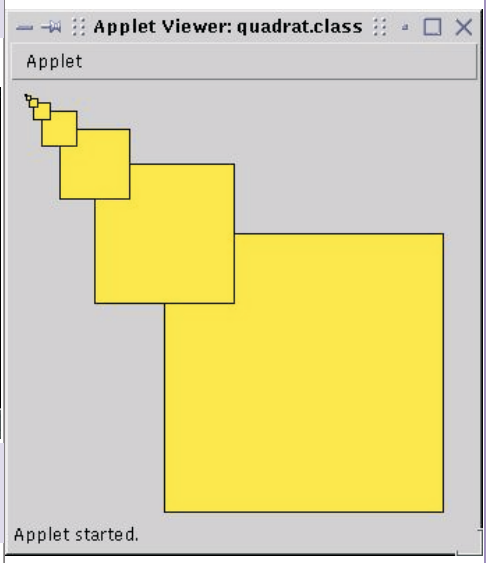
Beispiel: Flocke (flocke.java)




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

public class flocke extends Applet {
  public void paint (Graphics g) {
    Turtle t = new Turtle(this, 200,200);
    flockez(t,130);
  }

  public void flockez(Turtle t, double stamm)
  {
      for (int i=1;i<=6;i++) {
          t.rt(60);
          t.fd(stamm);
          if (stamm>2) flockez(t, stamm/3);
          t.fd(-stamm);
      }
  }
}
```

In dieser Klasse wird die Methode `flockez()` innerhalb der for-Schleife sechsmal aufgerufen. Davor wird der Winkel jeweils um 60° erhöht. Überlege, wie die Zahl der »Äste« der Flocke erhöht oder verringert werden könnte! Wirkt sich die Änderung der Zahl der Äste auch auf die rekursiven Strukturen aus?

Beispiel: Quadrate (quadrat.java)




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

public class quadrat extends Applet {

  public void paint (Graphics g) {
    qu(this, 100,210,210);
  }

  public void qu(Container ct, double l,
     double x, double y) {
    Graphics g = ct.getGraphics();
    if (l>0.5) {
      g.setColor(Color.black);
      g.drawRect((int) (x-l), (int) (y-l),
          (int) (2*l), (int) (2*l));
      g.setColor(Color.yellow);
      g.fillRect((int) (x-l+1),
          (int) (y-l+1),
          (int) (2*l-1),
          (int) (2*l-1));
      qu(ct, l/2, x-l, y-l);
    }
  }
}
```

Bemerkung: Außer den hier dargestellten rekursiv erzeugten Fraktalen gibt es eine zahlreiche Fraktale, die relativ einfach iterativ hergestellt werden können. Zwei davon sollen in einem späteren Beitrag behandelt werden: das **Apfelmännchen** und die **Julia-Menge**.