7 Rekursionen

Einführung

Die Rekursion ist ein Bauprinzip, das in vielen Dingen steckt. Wir lernen es hier kennen und experimentieren dann damit in der Igelgrafik. Es führt uns bis zu den Fraktalen. In der deutschen Sprache ist es leicht Rekursionen, die die Eigenschaft, Schachtelsätze bilden zu können, ausnutzen, zu veranschaulichen: Man kann zwischen Kommas Nebensätze einfügen. Wird die Tiefe der Schachtelung größer, werden sie leicht unübersichtlich. Fehlt die Abbruchbedingung in einem rekursiven Programm, so erhalten wir eine Endlosschleife und der Rechner „hängt sich auf". (Man betrachte sich einmal zwischen zwei parallelen Spiegeln.)

Beispiel 1: Spiegelfechterei:

```
import java.applet.*; import java.awt.*;
class Reku1 extends Applet{
      Label l; String s;
      Reku1(){
            s="Vor dem Spiegel: Es war einmal ein Mann";
            l=new Label(s); add(l);
            s=", der sich rasierte ";
            l=new Label(s); add(l);
      }
}
```

Zwischen die beiden Satzteile "Es war einmal ein Mann" und ", der sich rasierte " fügen wir nun weitere Nebensätze mit der rekursiven Methode rekutext() ein und zeigen sie an:

```
import java.applet.*; import java.awt.*;
class Reku2 extends Applet{
      String text=""; String s; Label l;
      Reku2(){
            s="Vor dem Spiegel: Es war einmal ein Mann";
            l=new Label(s); add(l);
            s=Rekutext(3);
            l=new Label(s); add(l);
            s=", der sich rasierte ";
            l=new Label(s); add(l);
      }
      String rekutext(int m){
            if (m>0){
                  text=text + ", der einen Mann sah " + m;
                  rekutext(m-1);
                  text=text + ", der sich rasierte " + m;
            }
            return text;
      }
}
```

In der Methode *rekutext(int m)* ist das Bemerkenswerte der **Selbstaufruf** *rekutext(m-1)*. Die Zahl m zählt dabei die Zahl der Wiederholungen mit, damit wir den Überblick behalten. Beim Ausdruck des Ergebnisses stellen wir mit "Aha" fest, dass nicht nur die Anweisungen bis zum Selbstaufruf wiederholt werden, sondern dass auch der Rest des Satzes ausgegeben wird:

Vor dem Spiegel: Es war einmal ein Mann,
der einen Mann sah (3), der einen Mann sah (2), der einen Mann sah (1),
der sich rasierte (1), der sich rasierte (2), der sich rasierte (3),
der sich rasierte.

Wenn man schrittweise tiefer in den Keller geht, muss man die gleiche Anzahl Stufen wieder hoch!

Beispiel 2: Fakultät

Viele mathematische Zusammenhänge werden durch rekursive Vorschriften beschrieben. Sie lassen sich elegant in Programmcode übertragen, in dem ein Selbstaufruf vorkommt. Die folgende Klasse zur Berechnung von Fakultäten enthält wieder eine rekursive Methode. Für ganze Zahlen größer als 1 ist n Fakultät

n! = n * (n-1) * ... * 1

n! wird gleich 1 gesetzt, wenn n=0, sonst wird n mit der Fakultät von n-1, also mit (n-1)! multipliziert:

```
import java.applet.*; import java.awt.*;
public class Faku extends Applet{
int n=4;
public Faku(){
      int y=f(n);
      Label l=new Label(""+y); add(l);
}
public int f(int n){
      if (n==0) return 1;
      else return n*f(n-1);   // Selbstaufruf
}
}
```

Verfolgen wir den Ablauf und steigen in den Keller! Im Konstruktor *Faku()* wird die Methode *f* mit dem Eingabeparameter *n=4* aufgerufen: *y=f(n).*

4! ist 4*3!	(1. Ebene) in Java formuliert: *n*f(n-1);* Die Multiplikation wird zunächst zurück gestellt und *f* noch mal aufgerufen, diesmal mit der Eingabe 3. n merkt man sich.
3! ist aber 3*2!	(2. Ebene) *f* wird mit der Eingabe 2 aufgerufen, n merkt man sich.
2! ist aber 2*1!	(3. Ebene) *f* wird mit der Eingabe 1 aufgerufen, n merkt man sich.
1! ist aber 1*0!	(4. Ebene) *f* wird mit der Eingabe 0 aufgerufen. n merkt man sich.
0! ist aber 1.	(5. Ebene) if (n==0) return 1; kein Selbstaufruf mehr.

Nun wieder zurück an die frische Luft. Die einzelnen Multiplikationen mit dem gespeicherten n werden dabei ausgeführt. Im Arbeitsspeicher ist jeweils eine Kopie der Methode `f` vorhanden, wenn sie noch nicht vollständig abgearbeitet wurde.

1*1=1	(4.Ebene)
1*1*2=2	(3.Ebene)
1*1*2*3=6	(2. Ebene)
1*1*2*3*4=24	(1. Ebene)

Bemerkung: Dieses Problem lässt sich auch nicht-rekursiv lösen. (Man beachte außerdem, dass die Werte der Fakultätsfunktion schnell den Bereich der darstellbaren Zahlen überschreiten.)

Beispiel 3, Euklids Satz: "Es gibt unendlich viele Primzahlen" lässt sich so beweisen:
Wir nehmen eine Zahl Z und bilden die Fakultät Z! Dazu addieren wir 1. Die Zahl Z!+1 ist offensichtlich nicht durch 2 teilbar, da beim Dividieren durch 2 der Rest 1 bleibt. Sie ist nicht durch 3, nicht durch 4, und so weiter, nicht durch Z teilbar, da immer der Rest 1 bleibt. Zu jeder Zahl Z gibt es also eine Zahl, die größer als Z ist und entweder Primzahl ist oder, wenn sie keine Primzahl ist, ist sie durch eine Primzahl teilbar, die größer als Z ist. Es gibt also eine größere Primzahl als Z. Das gilt für beliebiges Z.

Ein Beispiel für Rekursionen wird von **Hofstadter** so beschrieben:
Ein Manager besitzt ein besonders raffiniertes Telefon, auf dem er viele Anrufe empfängt. Wenn er gerade telefoniert, was er so gut wie immer tut, empfängt er einen Anruf von einem noch wichtigeren Manager. Er unterbricht sein Gespräch deshalb vorläufig, um sich dem zweiten zu widmen. Da kommt ein Anruf von noch weiter oben und es geschieht das Gleiche...

Oder: Ein Mops lief in die Küche und stahl dem Koch ein Ei. Da nahm der Koch den Löffel und schlug den Mops zu Brei. Da kamen viele Möpse und gruben ihm ein Grab. Sie setzten einen Grabstein, worauf geschrieben stand: Ein Mops lief in die Küche ...

⇨ Aufgabe 71: Schreiben Sie eine Klasse Mops, die sich selbst aufruft.

7.1 Igelgrafik

Ein einfacher Igel in Java

Rekursionen lassen sich in der Igelgeometrie am elegantesten veranschaulichen, denn Selbstaufrufe führen zu reizvollen selbstähnlichen Figuren. Große Formen wiederholen sich im Kleinen.
Was heißt zeichnen mit dem Igel? Der Begriff kommt aus dem Englischen, von Turtle-Grafik (turtle = Schildkröte). Die Programmiersprache LOGO war ein Vorreiter. Schon im Kindergarten gaben die Kinder der LOGO-Schildkröte Befehle wie "`vor 50`" oder "`rechts 90`". Sie geht dann 50 Pixel vor, dreht sich um die eigene Achse um 90° und hinterlässt ihre Spur als Strich auf dem Bildschirm:
Durch das LOGO-Programm

```
vor 50 rechts 120 vor 50 rechts 120 vor 50 rechts 120
```

entsteht ein gleichseitiges Dreieck.
Wiederholungen werden rekursiv geschrieben. Ohne Abbruchbedingung, also ohne Ende springt der Igel beim folgenden LOGO-Programm im Quadrat:

```
Pr Quadrat
     vor 50
     rechts 90
     Quadrat
```

Realisieren wir einen eigenen Igel in Java, also ein Objekt, das zeichnen kann und über Methoden wie *vor(...)* und *rechts(...)* verfügt! Im folgenden Programm geht der Igel I auf Knopfdruck einfach nur 10 Pixel vor:

```
import java.applet.*; import java.awt.*; import java.awt.event.*;
public class Igel0 extends Applet implements ActionListener{
     Igel i= new Igel();
     public Igel0() {
          i.setSize(400,400); i.setBackground(Color.yellow); add(i);
          Button b = new Button("zeichne!"); add(b);
          b.addActionListener(this);
     }
     public void actionPerformed(ActionEvent e){
          i.vor(10);
     }
public class Igel extends Canvas {
     int xa=200; int ya=200;
     public void vor(int d) {
          Graphics stift=getGraphics();
          int ye=ya-d;
          stift.drawLine(xa,ya,xa,ye);
          ya=ye;
     }
}}
```

Der Igelbereich hat hier die Größe 400 x 400 Pixel und einen gelben Hintergrund. Beim Drücken des Buttons B geht der Igel 10 Pixel vorwärts. Die Klasse Igel ist von der Klasse Canvas (engl. Leinwand) abgeleitet. Sie stellt die Grafik-Befehle des Pakets Graphics zur Verfügung. x_a und y_a sind die Anfangskoordinaten, x_e und y_e die Endkoordinatern des Igelweges.

⇨ Aufgabe 72: Schreiben Sie eine Methode *rechts()*. Der Igel soll sich um seine eigene Achse drehen können und in die Richtung, in die er schaut, gehen. Die Klasse Igel braucht also als Daten die Position (x,y) und den Winkel relativ zur Nordrichtung. Als Methoden werden benötigt: 1. vor mit der Angabe, wieweit und 2. rechts mit der Angabe um welchen Winkel gedreht werden soll. Das Winkelmaß wandeln wir um vom Gradmaß ins Bogenmaß, da die Sinusfunktion ihr Argument im Bogenmaß verlangt.

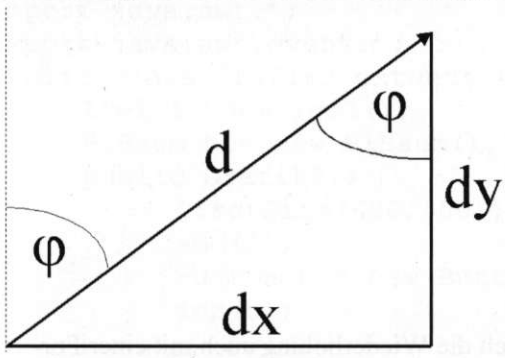
Igel- Figuren

Hier ein Igel, der sich auch drehen kann:

```
import java.applet.*;
import java.awt.*;
public class Igel extends Canvas {
     int xa=300; int ya=300;
     int phi=0;
     double b=0;
     public void vor(int d) {
          Graphics stift=getGraphics();
          int dx=(int)(d*Math.sin(b));
          int dy=(int)(d*Math.cos(b));
          int xe=xa+dx; int ye=ya-dy;
          stift.drawLine(xa,ya,xe,ye);
          xa=xe; ya=ye;
     }
     public void rechts(int Winkel) {
          phi=phi+Winkel;
          b=2*Math.PI*phi/360;

     }
}
```

Zur Erklärung: Anfangs wird ein Punkt in der Mitte der Zeichenfläche (bei Pixel 300/300) als Ausgangspunkt gewählt. Wenn die Eingabe d, die Distanz, um die der Igel vorwärts gehen soll, gegeben ist, berechnen wir die Koordinaten des Endpunktes, indem wir dx nach rechts und dy nach oben gehen. dx und dy berechnen wir im rechtwinkligen Dreieck (siehe Abbildung). Es gilt



$$\sin \varphi = dx / d$$
$$dx = d \sin \varphi$$

Die Sinusfunktion aus dem Mathematikpaket Math benötigt allerdings den Winkel im Bogenmaß. Für das Bogenmaß b gilt:

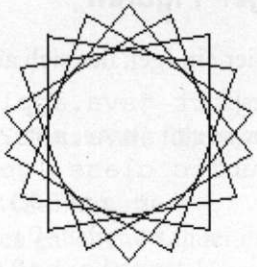
$$b / 2\pi = \varphi / 360°$$
$$b = \pi \varphi / 180°$$

Ist die Verbindungslinie zwischen Anfangs-und Endpunkt gezogen, so werden die Koordinaten des Endpunktes xe und ye als neue Anfangskoordinaten gespeichert.
Der Igel soll sich auf der Stelle drehen können. Wenn φ die Richtung ist, in die der Igel schaut (immer bezogen auf die Nordrichtung, also auf dem Bildschirm nach oben), dann müssen wir den Drehwinkel einfach addieren. Wird in der Methode *rechts(...)* eine negative Zahl eingegeben, so bedeutet dies eine Drehung nach links.
Dieser Igel ist in vielen Programmen wieder verwendbar. Am besten wir speichern den Igel zusammen mit den Programmen, die ihn aufrufen wollen in ein extra Verzeichnis. Dazu wird die Klasse Igel compiliert und kann dann zum Beispiel von der Klasse Figur2 benutzt werden, da sie im gleichen Verzeichnis liegt. Das Verzeichnis enthält dann zwei Dateien mit der Endung *.class und die aufrufende HTML-Datei.

Im folgenden Programm geht der Igel jedes Mal um 100 Pixel vor und dreht sich um 100°, wenn man einen Button anklickt. Die Igelmethoden *vor* und *rechts* werden in actionPerformed aufgerufen:

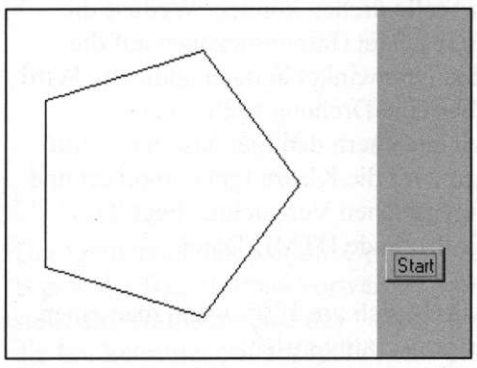
```
import java.applet.*; import java.awt.*;
import java.awt.event.*;
public class Figur2 extends Applet implements ActionListener{
     Igel i= new Igel();
     public Figur2() {
          i.setSize(400,400); add(i);
```

```
        Button b = new Button("zeichne!");
        b.addActionListener(this);
        add(b);
    }
    public void actionPerformed(ActionEvent e){
        i.vor(100); i.rechts(100);
    }
}
```



Im nächsten Programm ist eine Klasse Quadrat rekursiv definiert. Ihre Methode `zeichne` bekommt die Anzahl der Strecken als Argument. Die Klasse Quadrat enthält einen Selbstaufruf. Dabei wird n immer um 1 reduziert, bis es Null wird. Im Hauptspeicher des Rechners wird jedesmal beim Selbstaufruf eine Kopie der Methode zeichne angelegt, in der die Anweisungen noch mal abgearbeitet werden. Es wiederholt sich 4 mal, dass der Igel 50 Pixel vorwärts geht und sich um 90 Grad dreht:

```
import java.applet.*; import java.awt.*; import java.awt.event.*;
public class Grafik1 extends Applet implements ActionListener{
    Igel i= new Igel();
    Quadrat qu= new Quadrat();
    public Grafik1() {
        i.setSize(400,400); add(i);
        Button b = new Button("Start");
        b.addActionListener(this);
        add(b);
    }
    public void actionPerformed(ActionEvent e){
            qu.zeichne(4);
    }
class Quadrat{
    public void zeichne(int n){
        if (n>0){
            i.vor(50); i.rechts(90);
            zeichne(n-1);
        }
    }
}
}
```



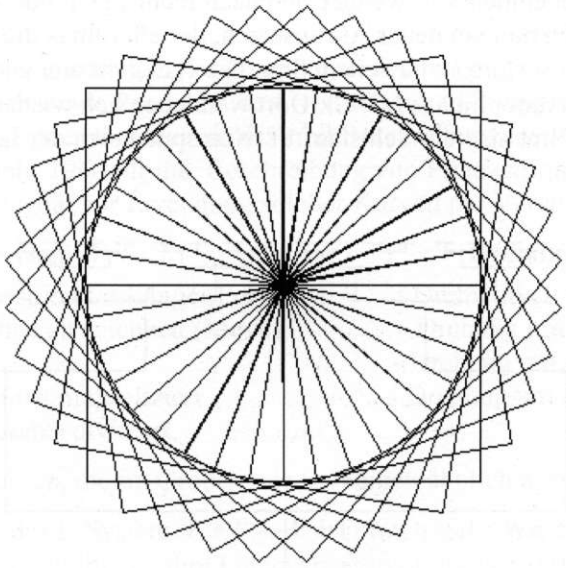
Nicht-rekursiv lässt sich die Wiederholung auch mit einer For-Schleife bewirken:

```
class Vieleck{
   public void zeichne(int n){
      for (int k=0; k<n; k=k+1){
            i.vor(50); i.rechts(360/n);
      }
   }
}
```

In der Abbildung erfolgte der Aufruf mit n = 5.

⇨ Aufgabe 73: Ändern Sie die Klasse Quadrat so, dass anstatt der 4 die Seitenlänge angegeben werden kann. Falls keine Seitenlänge angegeben wird, sollen 100 Pixel angenommen werden.

⇨ Aufgabe 74: Definieren Sie eine Klasse VielQuadrat, so dass viele Quadrate erzeugt werden können. Sie sollen jeweils um beispielsweise 10° gedreht sein:

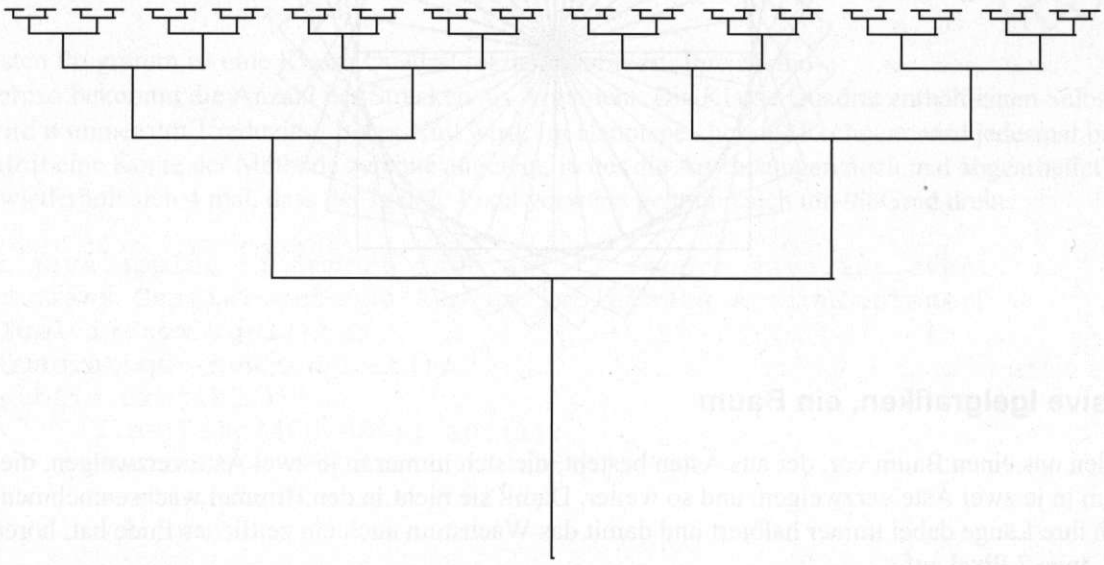


Rekursive Igelgrafiken, ein Baum

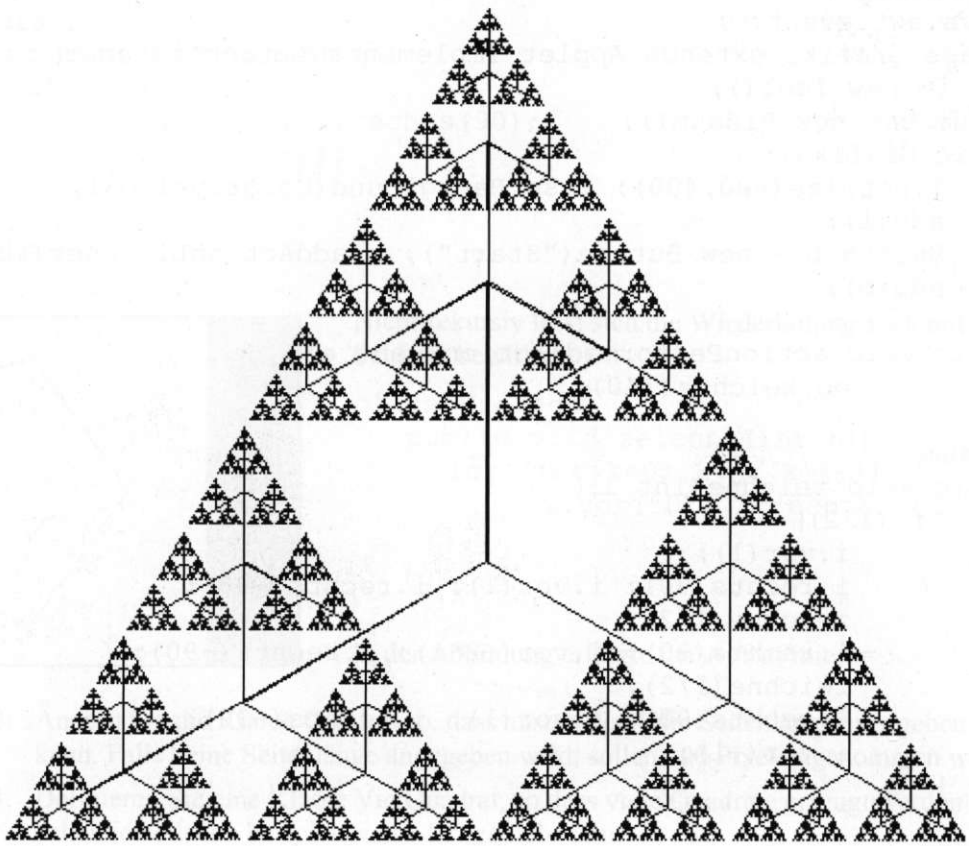
Wir stellen uns einen Baum vor, der aus Ästen besteht, die sich immer in je zwei Äste verzweigen, die sich wiederum in je zwei Äste verzweigen, und so weiter. Damit sie nicht in den Himmel wachsen nehmen wir an, dass sich ihre Länge dabei immer halbiert und damit das Wachstum auch ein zeitliches Ende hat, hören wir bei der Länge 2 Pixel auf.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Grafik3 extends Applet implements ActionListener{
      Igel i= new Igel();
      BiBaum bb= new BiBaum();
      public Grafik3() {
            i.setSize(600,400); i.setBackground(Color.yellow);
            add(i);
            Button b = new Button("Start"); b.addActionListener(this);
            add(b);
      }
      public void actionPerformed(ActionEvent e){
                  bb.zeichne(140);
      }
class BiBaum{
      public void zeichne(int l){
            if (l>2){
                  i.vor(l);
                  i.rechts(90); i.vor(l); i.rechts(-90);
                  zeichne(l/2);
                  i.rechts(90); i.vor(-2*l); i.rechts(-90);
                  zeichne(l/2);
                  i.rechts(90); i.vor(l); i.rechts(-90);
                  i.vor(-l);
            }
      }
}
}
```

Der Igel geht zuerst eine Längeneinheit vor, wendet sich nach rechts, geht vor, dreht sich (-90) und schaut nach oben. An dieser Stelle kann nun ein neuer Ast ansetzen, der allerdings die Länge L/2 hat. Dies wird durch den Selbstaufruf `zeichne(L/2)` bewerkstelligt. Der Igel geht nun wieder zurück bis zur Abzweigung und die gleiche Strecke darüber hinaus (-2*L). Dort wendet er sich wieder nach oben. Da hier wieder ein neuer Ast ansetzen soll, kommt also ein Selbstaufruf. Nun sputet sich der Igel, um schleunigst wieder zum Ausgangspunkt zurückzukehren.



Ein ternärer Baum hat immer drei statt zwei Verzweigungen. Es entsteht ein Dreieck, aus dem Dreiecke ausgeschnitten sind, das sogenannte Sierpinski-Dreieck:



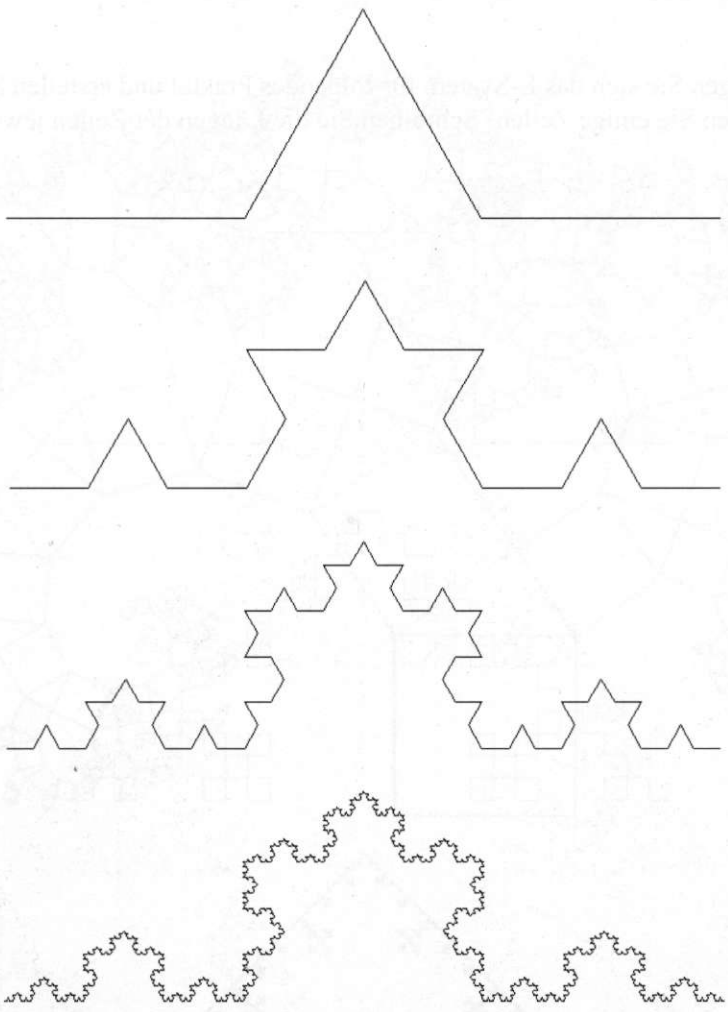
7.2 Fraktale

Es gibt Kurven mit der Eigenschaft, unendlich lang zu sein, aber eine endliche Fläche einzuschließen. Diese Kurven wurden von Mathematikern wie Peano, Hilbert, von Koch und Sierpinski untersucht. Die Kurven lassen sich durch sogenannte L-Systeme erzeugen, das sind Beschreibungen, die der dänische Biologe Aristid Lindenmayer (1925-1989) im Jahr 1968 für die Beschreibung von Pflanzenformen eingeführt hat Ein L-System besteht aus folgenden Regeln zur Erzeugung solcher Kurven per Vereinbarung:

Zeichnen von Strecken	**Drehungen**
F Zeichnen einer Linie bestimmter Länge (Forward)	a Drehwinkel
M Vorwärtsbewegung ohne zu zeichnen (Move)	+ Drehung um a entgegen dem Uhrzeigersinn
s Streckungsfaktor	- Drehung um a im Uhrzeigersinn
* Die Linienlänge wird mit s multipliziert	| Drehung um 180°
: die Linienlänge wird durch s dividiert	

Selbstähnliche Kurven, also Kurven, die im Groben wie im Feinen ähnlich aussehen, heißen **Fraktale** [10].

Wir erzeugen ein Fraktal durch das L-System F + F - - F + F mit a=60°. Wir ersetzen jede Strecke nach der Regel Regel F → F + F - - F + F, dividieren ihre Länge aber zuvor durch 3. Das wiederholen wir und erreichen eine immer feinere Zergliederung. Die entstehende Kurve heißt auch Koch-Kurve nach dem Mathematiker Koch:

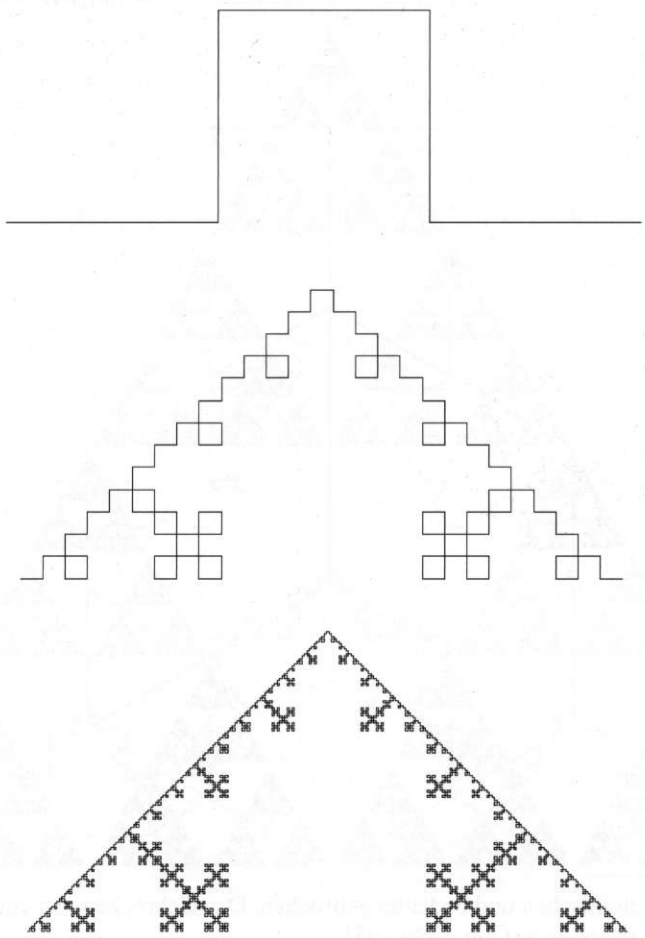


[10] Der Name kommt aus dem Lateinischen und bedeutet gebrochen. Das Gebrochene an einem Fraktal ist seine Dimension, wir gehen hier jedoch nicht näher darauf ein, siehe [93].

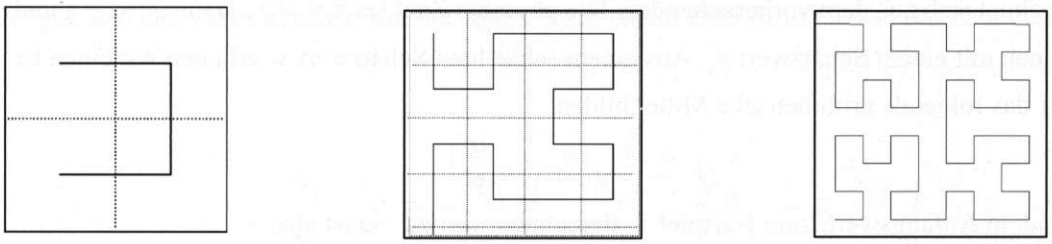
Programmcode für die Koch-Kurve:

```
import java.applet.*; import java.awt.*; import java.awt.event.*;
public class Grafik4 extends Applet implements ActionListener{
      Igel i= new Igel();
      KochKurve kk= new KochKurve();
      public Grafik4() {
         i.setSize(600,300); i.setBackground(Color.yellow); add(i);
         Button b= new Button("Start"); b.addActionListener(this); add(b);
      }
      public void actionPerformed(ActionEvent e){
            i.rechts(90); k.zeichne(600);
      }
class KochKurve{
      public void zeichne(int l){
            if (l<20) I.vor(l);
            else {
                  l=l/3;
                  zeichne(l); i.rechts(-60); zeichne(l); i.rechts(120);
                  zeichne(l); i.rechts(-60); zeichne(l);
            }
      }
}}
```

⇨ Aufgabe 75: Überlegen Sie sich das L-System für folgendes Fraktal und erstellen Sie das Java-Programm: Ergänzen Sie einige Zeilen! Schreiben Sie die Längen der Zeilen jeweils daneben! Was fällt auf?



Peano und Hilbert untersuchten Kurven, die eine Ebene füllen. Ein Quadrat wird in vier Teile unterteilt und jedes Teil von der Kurve besucht. Unterteilt man wieder jedes Teilquadrat, so hat die Kurve immer mehr Punkte zu besuchen. Je feiner man die Unterteilung macht, desto besser füllt die Kurve die Ebene. Beim Abtasten eines Grautonbildes kann man nun dieses Prinzip anwenden. Ein Drucker, der nur schwarz oder weiß drucken kann, druckt bestimmte Punktmuster. Diese Technik wird Dithering genannt. Tastet man ein Bild in quadratischen Blöcken ab, so entstehen bei geraden Linien unerwünschte Muster. Die kann man vermeiden, indem man die Ebene beispielsweise entlang einer solchen Hilbertkurve abtastet:



⇨ Aufgabe 76: Zeichnen Sie ein Quadrat und darauf ein rechtwinkliges Dreieck und auf jeder Kathete ein Quadrat und darauf ein rechtwinkliges Dreieck, und so weiter. Es wird behauptet es entstünde eine Broccoli-ähnliche Figur. Sie heißt auch Pythagoras-Baum.