3.9 Koordinatensysteme

Sehr oft werden auf dem Bildschirm Graphen gezeichnet, deren Punkte mit Hilfe von Zuordnungsvorschriften berechnet werden. Meist handelt es sich dabei um *Funktionsgraphen*, wie sie aus dem Mathematikunterricht gut bekannt sind. Dabei treten immer wieder die gleichen Probleme auf:

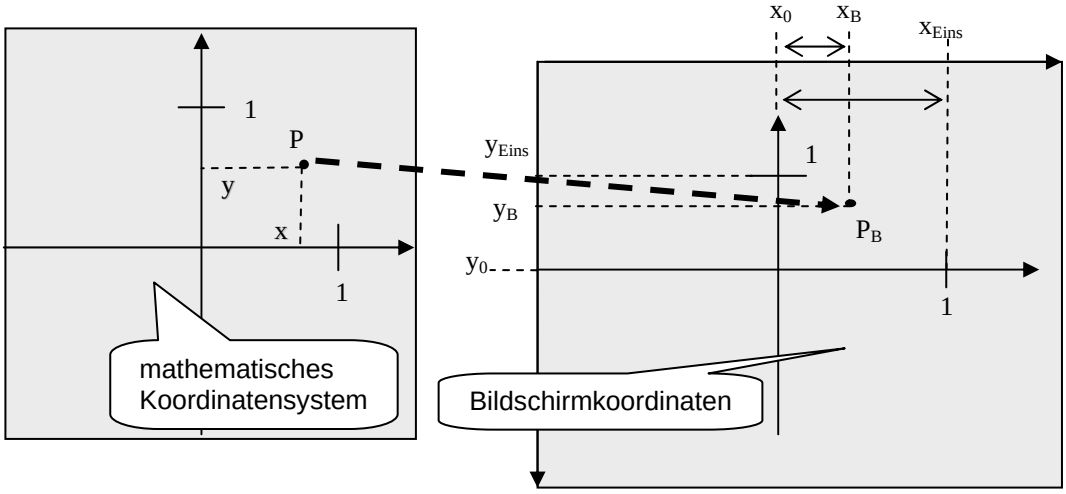
- Punkte des „mathematischen“ Koordinatensystems müssen auf „Bildschirmkoordinaten“ umgerechnet werden.
- Zahlenwerte müssen geändert, ggf. vom Benutzer eingegeben werden. Dabei sollten eventuelle Fehler abgefangen werden.
- die geänderten Graphen müssen am Bildschirm dargestellt werden.

Wir wollen diese Probleme nacheinander lösen und in einigen Anwendungen einsetzen.

3.9.1 Koordinatentransformationen

Mathematische und Bildschirmkoordinaten unterscheiden sich in drei Punkten:

- Der *Ursprung*, also der Punkt (0|0), liegt beim mathematische Koordinatensystem meist in der Mitte des beschriebenen Blattes; auf dem Bildschirm befindet er sich in der linken oberen Ecke.
- Die *y-Achse* ist bei mathematischen Koordinaten nach oben gerichtet, auf dem Bildschirm aber nach unten.
- Die *Längeneinheit* ist im mathematischen Koordinatensystem durch das Intervall von Null bis Eins gegeben, auf dem Bildschirm beträgt sie ein Pixel (einen Bildpunkt).



Trotz aller Unterschiede sollen die erzeugten Graphen in beiden Koordinatensystemen ähnlich sein, d. h. sich entsprechende Strecken sollten im gleichen Verhältnis zueinander stehen. Dieses Ziel nutzen wir aus. Wir fordern:

> *Die Strecke x vom Nullpunkt des mathematischen Koordinatensystems zur x-Koordinate eines Punktes verhält sich zur Längeneinheit 1 dieser Achse genauso, wie die entsprechenden Strecken auf dem Bildschirm. Für die y-Koordinaten gilt Entsprechendes.*

Bezeichnen wir (wie in der Zeichnung angegeben) die Bildschirmkoordinaten des Ursprungs des auf den Bildschirm abgebildeten mathematischen Koordinatensystems als x_0 bzw. y_0 und die Bildschirmkoordinaten der Punkte, die einer Längeneinheit auf den Achsen entsprechen, mit x_{Eins} bzw. y_{Eins}, dann erhält man die Bildschirmkoordinaten x_B bzw. y_B eines Punktes $P(x|y)$, indem man die entsprechenden Verhältnisgleichungen aufstellt:

$$\frac{x}{1} = \frac{x_B - x_0}{x_{Eins} - x_0} \qquad \frac{y}{1} = \frac{y_B - y_0}{y_{Eins} - y_0}$$

Diese Beziehungen können wir je nach Bedarf entweder nach x und y auflösen (wenn wir die Bildschirmkoordinaten kennen und die entsprechenden mathematischen Koordinaten berechnen wollen), oder nach x_B und y_B, wenn wir aus den mathematischen die Bildschirmkoordinaten berechnen. Es ergibt sich:

$$x = \frac{x_B - x_0}{x_{Eins} - x_0} \qquad x_B = x \cdot (x_{Eins} - x_0) + x_0$$

$$y = \frac{y_B - y_0}{y_{Eins} - y_0} \qquad y_B = y \cdot (y_{Eins} - y_0) + y_0$$

3.9.2 Zahleneingabe mit einem Textfeld

Mit Hilfe von *Textfeldern*, also Komponenten vom Typ *TextField*, können einzeilige Texte auf dem Bildschirm dargestellt und vom Benutzer verändert werden, wenn das vorgesehen ist (die Eigenschaft *editable* muss dann auf *true* stehen). Der aktuelle Text des Textfeldes ist in der Eigenschaft *text* enthalten. Heißt also z. B. ein Textfeld *tfYEins*, dann kann der aktuelle Wert mithilfe des Aufrufs von *tfYEins.getText()* gelesen werden. In Textfeldern können wir aber nur Zeichenketten speichern, d. h. wir müssen unsere Zahlen bei Bedarf in Zeichenketten verwandeln – und umgekehrt. Zu diesem Zweck stehen verschiedene *Typwandlungsfunktionen* zur Verfügung:

- *Integer(zahl).toString()* verwandelt eine ganze Zahl in einen String. Etwas einfacher lässt sich das Gleiche erreichen, wenn man der Zahl einen leeren String voranstellt: (““+*zahl*). Wir werden meist diese Methode verwenden.
- *Integer.parseInt(string)* bestimmt die in einem String enthaltene ganze Zahl.
- *Double.parseDouble(string)* bestimmt die in einem String enthaltene Gleitpunktzahl.

Üblicherweise wird beim Programmentwurf dem benutzten Textfeld ein Anfangswert mit Hilfe des Objektinspektors zugewiesen. Wird dieser Wert während des Programmlaufs geändert und die ENTER-Taste gedrückt, dann tritt das *actionPerformed*-Ereignis des Textfeldes ein. In der zugeordneten Ereignisbehandlungsprozedur wird dann die neue Zahl bestimmt, z. B. durch

```
xEins = Integer.parseInt(textField.getText());
```

Was passiert aber, wenn der Benutzer einen Fehler macht, also eine Zeichenkette eingibt, die gar keine Zahl enthält. Unser Programm würde mit einer Fehlermeldung abbrechen – und das wollen wir natürlich verhindern.

Wir benutzen deshalb einen *try...catch...*-Block, in dem wir versuchen, die in einem String enthaltene Zahl zu bestimmen. Tritt dabei ein Fehler (eine *NumberFormatException*) auf, dann werden die im *catch*-Teil vorgesehenen Anweisungen ausgeführt. Damit lautet die vollständige Ereignisbehandlungsroutine:

```
public void actionPerformed(java.awt.event.ActionEvent e)
{
  try
  {
    xEins = Integer.parseInt(textField.getText());
  }
  catch(NumberFormatException ex)
  {
    textField.setText("Eingabefehler!");
  }
}
```

3.9.3 Zahleneingabe mit einem Schieberegler

Soll sich ein Wert nur in einem vorgegebenen Zahlenbereich ändern können, dann kann diese Änderung mit Hilfe einer *Scrollbar*-Komponenten erfolgen, die wir z. B. von den Rändern eines Windows-Fensters her kennen, in dem sich mehr befindet, als bei der momentanen Fenstergröße dargestellt werden kann. Komfortabel wird so eine Zahleneingabe, wenn der eingestellte Wert direkt in einem Textfeld angezeigt wird. (Beschränken wir uns auf die reine Anzeige, dann sollte dessen *editable*-Eigenschaft auf *false* gesetzt werden.) Da die so bestimmten Zahlen im Programm berechnet werden, kann auch keine fehlerhafte Eingabe erfolgen, so dass wir uns einen *try...catch...*-Block ersparen.

Der Wert einer Scrollbar-Komponente kann sich standardmäßig zwischen den Werten 0 und 100 bewegen. (Das kann im Objektinspektor auch abgeändert werden.) Die momentane Position des Schiebereglers erhält man mit der *getValue*-Methode. Aus deren Wert und dem Maximalwert kann dann leicht der gewünschte Wert abgeleitet werden. Soll sich z. B. eine Variable *x* mit Hilfe einer Scrollbar *sbx* zwischen den Werten *1* und *2,5* ändern, dann erhalten wir[14]:

```
x = 1 + 1.5*sbx.getValue()/100;
```

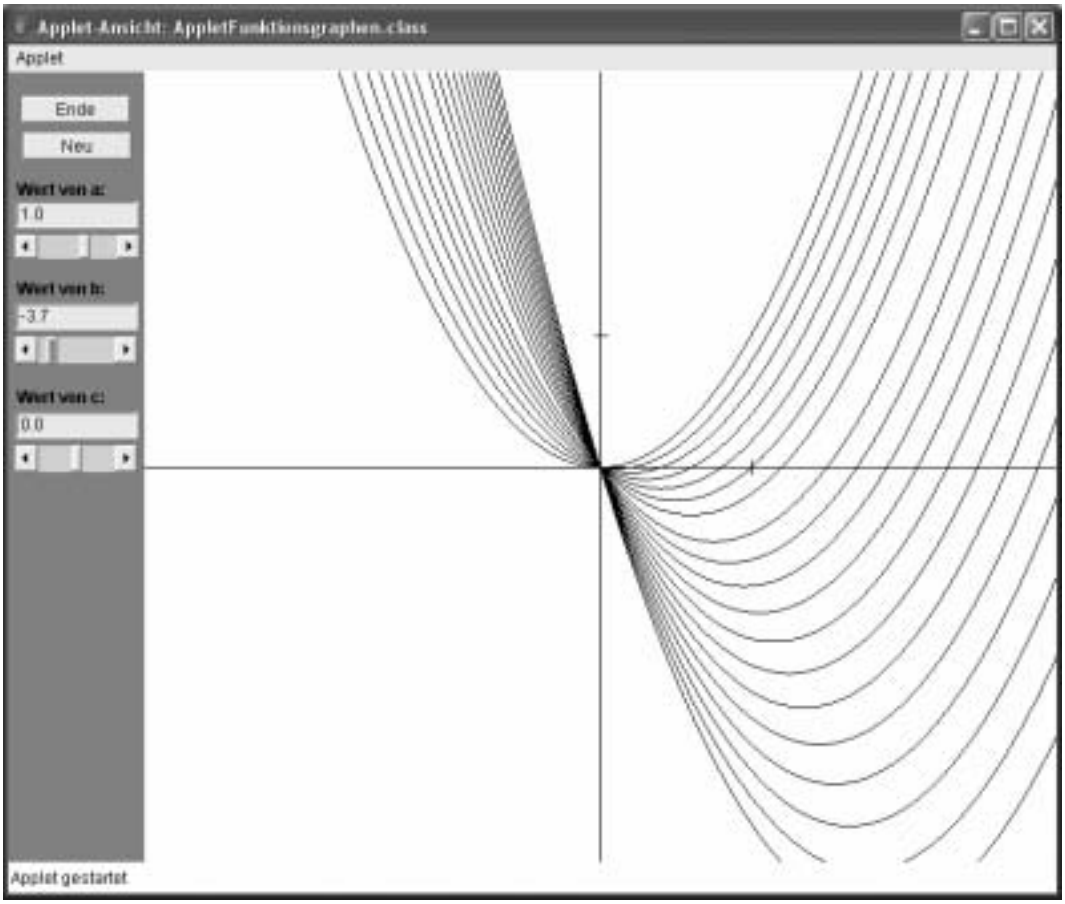
Bewegt man den Schieberegler mit der Maus, dann wird ein *AdjustmentValueChanged*-Ereignis ausgelöst, auf das mit einer Ereignisbehandlungsroutine reagiert wird. In dieser wird eine Zahl *x* bestimmt und im Textfeld dargestellt:

```
public void adjustmentValueChanged
                              (java.awt.event.AdjustmentEvent e)
{
  Double x = 1+1.5*sbx.getValue()/100;
  tfEinheit.setText(""+x);
}
```

[14] Das gilt nur, falls sich mithilfe des Schiebereglers wirklich der Maximalwert (hier: 100) einstellen läßt. Ansonsten muss man die Werte anpassen.

3.9.4 Funktionsgraphen

Wir wollen unsere Kenntnisse jetzt auf ein Standardgebiet der Schulmathematik anwenden: das Zeichnen der Funktionsgraphen von Parabeln der Form $y = a \cdot x^2 + b \cdot x + c$. Dazu erzeugen wir eine *Panel*-Komponente, die drei Schieberegler mit Anzeigefeldern aufnimmt, mit deren Hilfe wir die Werte der Parameter *a, b* und *c* zwischen den Werten *−5* und *5* ändern können. Beim Applet stellen wir das *Borderlayout* ein und setzen im Panel die *constraint*-Eigenschaft auf *West*. Jetzt „klebt“ dieses Element am linken Rand des Fensters. Der Rest des Fensters dient als Zeichenfläche.



Immer wenn ein Parameter oder das Fenster geändert wurde, soll ein neuer Graph gezeichnet werden. Dazu schreiben wir eine Methode *fangeNeuAn*, in der die Anfangswerte gesetzt und die Achsen des Koordinatensystems neu gezeichnet werden.

```
private void fangeNeuAn()
{
  a = -5.0+10.0*sbA.getValue()/100;
  b = -5.0+10.0*sbB.getValue()/100;
  c = -5.0+10.0*sbC.getValue()/100;
  tfA.setText(""+a); tfB.setText(""+b); tfC.setText(""+c);
  x0 = (getWidth()-100)/2+100;
  y0 = getHeight()/2;
  xEins = x0 + (x0-100)/3; yEins = y0 - y0/3;
  Graphics g = getGraphics();
  g.setColor(Color.BLACK);
  g.drawLine(100,y0,2*x0,y0); g.drawLine(x0,0,x0,2*y0);
  g.drawLine(xEins,y0+5,xEins,y0-5);
  g.drawLine(x0-4,yEins,x0+5,yEins);
  xB = 100; repaint();
}
```

Parameterwerte aus den Schieberegler-stellungen bestimmen

Ursprung und Längeneinheiten berechnen

Koordinatensystem zeichnen

Dieses Unterprogramm wird in den entsprechenden Ereignisbehandlungsroutinen aufgerufen, nachdem die neuen Werte bestimmt sind. In der *paint*-Methode wird der Funktionswert in Abhängigkeit von der momentanen Bildschirmkoordinate *xB* berechnet. Danach wird eine entsprechende Linie gezeichnet und der Wert von *xB* erhöht und zum nächsten Punkt übergegangen. Anfangs wird geprüft, ob eine boolesche Variable *beimZeichnen* den Wert *true* hat. Ist dieses nicht der Fall, dann handelt es sich um den ersten Zeichenvorgang und *fangeNeuAn* wird aufgerufen.

```
public void paint(Graphics g)
{
  if(!beimZeichnen)
  {
    beimZeichnen = true;
    fangeNeuAn();
  }
  int xAlt=xB,yAlt;
  if(xB <= getWidth())
  {
    xB++;
    double x = 1.0*(xAlt-x0)/(xEins-x0);
    double y = a*x*x + b*x + c;
    yAlt = (int)Math.round(y*(yEins-y0)+y0);
    x = 1.0*(xB-x0)/(xEins-x0);
    y = a*x*x + b*x + c;
    yB = (int)Math.round(y*(yEins-y0)+y0);
    g.drawLine(xAlt,yAlt,xB,yB);
    repaint();
  }
}
```

Bei Bedarf Anfangswerte ermitteln und Koordinatensystem zeichnen

Alte Werte speichern

Nächsten Punkt wählen

mathematische Koordinaten

Umrechnung auf Bildschirmkoordinaten

Linie zeichnen – und auf ein Neues!

Zuletzt wird ein Button *Neu* eingeführt, dessen Ereignisbehandlungsmethode das gesamte Bild löscht.

```
public void actionPerformed(java.awt.event.ActionEvent e)
{
  Graphics g = getGraphics();
  g.setColor(Color.WHITE);
  g.fillRect(0,0,getWidth(),getHeight());
  fangeNeuAn();
}
```

Alles mit einem weißen Rechteck überschreiben …

... und neu beginnen.

Da alle Variable in unterschiedlichen Unterprogrammen benötigt werden, müssen sie global vereinbart werden.

```
double a,b,c;
int x0,y0,xEins,yEins,xB,yB;
boolean beimZeichnen = false;
```

3.9.5 Aufgaben

1. Schreiben Sie Programme, um die **Graphen**

   a: gebrochen rationaler Funktionen der Form $y = \frac{a \cdot x + b}{c \cdot x + d}$ auf dem Bildschirm darzustellen.

   b: von Potenzfunktionen der Form $y = a \cdot x^n$ darzustellen.

   c: von Exponentialfunktionen der Form $y = a \cdot e^x \cdot (b \cdot x^n + c \cdot x + d)$ darzustellen.

   d: von Logarithmusfunktionen darzustellen.

   Benutzen Sie bei Bedarf *try...catch...*-Blöcke zur Berechnung der Funktionswerte.

2. Verändern Sie das Programm zum **Graphenzeichen** so, dass

   a: die Skalen auf den Achsen verändert werden können.

   b: der dargestellte Bereich verschoben werden kann.

   c: durch Auswahl eines rechteckigen Bereichs mit der Maus automatisch gezoomt werden kann.

3. Schreiben Sie ein Programm, in dem eine **Wertetabelle** ausgefüllt werden kann, indem die Wertepaare in zwei Editierfelder eingegeben und auf Knopfdruck in eine Reihung (ein Feld) übernommen werden. Das Feld wird am Bildschirm dargestellt, indem in Texte verwandelte Zahlen geschrieben werden. Nach Abschluss der Eingabe soll ein entsprechender Graph gezeichnet werden.

4. a: Die Eckpunkte eines **Dreiecks** sollen in der Koordinatendarstellung erfragt werden: z. B. als A(-5|-3), B(6|1) und C(2|5). Dazu benötigt man sechs Editierfelder. Bei Bedarf kann die Zahleneingabe auch mit Hilfe von Schiebereglern erfolgen.

   b: Schon bei der Zahleneingabe sollen die Seitenlängen, die Winkel und die Fläche des Dreiecks berechnet und in Editierfeldern oder Label-Komponenten angezeigt werden. Mit Hilfe von *try...catch...*-Blöcken sollen Fehler abgefangen werden.

   c: Das Dreieck soll am Bildschirm dargestellt werden.

   d: Auf entsprechende Knopfdrücke sollen Umkreis, Inkreis, Mittelsenkrechte, Seitenhalbierende, ... gezeichnet werden.

5. a: Der ***Thaleskreis*** soll am Bildschirm dargestellt werden. Mit Hilfe eines Schiebereglers soll der Eckpunkt des Dreiecks auf dem Kreis verschoben werden können.

   b: Über den Seiten des Dreiecks sollen die entsprechenden Quadrate gezeichnet und ihre Flächen berechnet und angezeigt werden. Die Summe der Flächen der beiden kleineren Quadrate wird ebenfalls angezeigt. ***(Satz des Pythagoras***)

   c: Über der Höhe des Dreiecks sollen das Quadrat, über einem Höhenabschnitt das Rechteck mit der Seitenlänge des zweiten Höhenabschnitts gezeichnet und die Flächen berechnet und angezeigt werden. (***Höhensatz des Euklid)***

   d: Über einer Seite des Dreiecks soll das Quadrat, über der Hypotenuse das Rechteck mit der Seitenlänge des entsprechenden Höhenabschnitts gezeichnet und die Flächen berechnet und angezeigt werden. (***Kathetensatz des Euklid***).

6. a: Ein Kreis wird gezeichnet und in ihm eine Sehne. Die Enden der Sehne werden mit dem Kreismittelpunkt verbunden. Die Lage der Sehne kann mit Hilfe eines Schiebereglers verändert werden.

   b: Ein Punkt auf dem Kreisbogen über Sehne soll mit Hilfe eines Schiebereglers verschoben werden können. Der Punkt wird mit den Enden Sehne verbunden.

   c: Die Winkel in den entstandenen Dreiecken am Kreispunkt und am Mittelpunkt des Kreises sollen berechnet und angezeigt werden. (***Satz über die Peripheriewinkel***)

3.10 Simulationen und chaotisches Verhalten

3.10.1 Die Chaosparabel

Simuliert man mit Computern das Verhalten realer Systeme, dann geschieht das oft, indem man mithilfe bekannter Gesetzmäßigkeiten aus dem derzeitigen Zustand den darauf folgenden berechnet, und aus diesem wiederum einen späteren, usw. Man steckt also die Zwischenergebnisse wieder als Eingabewerte in eine Gleichung (oder ein Gleichungssystem), und berechnet daraus neue (Zwischen-)Ergebnisse. Eine typische Anwendung ist das Wetter von nächster Woche, für das aus den heute gemessenen Temperaturen, Luftdrucken, Niederschlägen, Wolkenverteilungen, ... das Wetter von morgen berechnet wird, und daraus das von übermorgen. Da Wettervorhersagen ziemlich ungenau sind, muss man sich fragen, woher diese Unsicherheit kommt, wenn doch genaue Messwerte und schnelle Supercomputer bereitstehen.

Bezeichnen wir den betrachteten Zustand zu einem Zeitpunkt als x_t und den darauf folgenden als x_{t+1}, dann berechnen wir x_{t+1} aus x_t mit Hilfe der Funktion f. Man spricht von einer *Iteration*.

$$x_{t+1} = f(x_t)$$

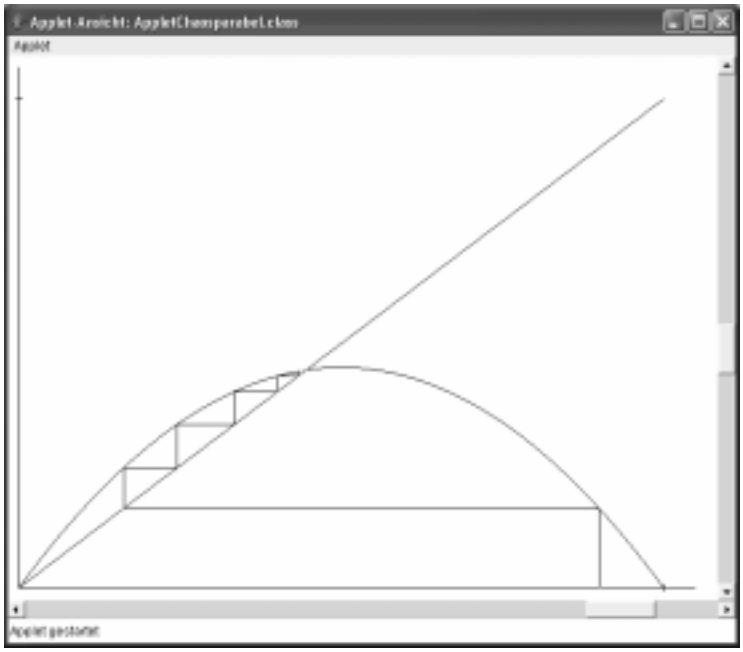
Eine sehr einfache solche Funktion wird durch die *logistische Gleichung*[15] beschrieben, eine einfache quadratische Funktion.

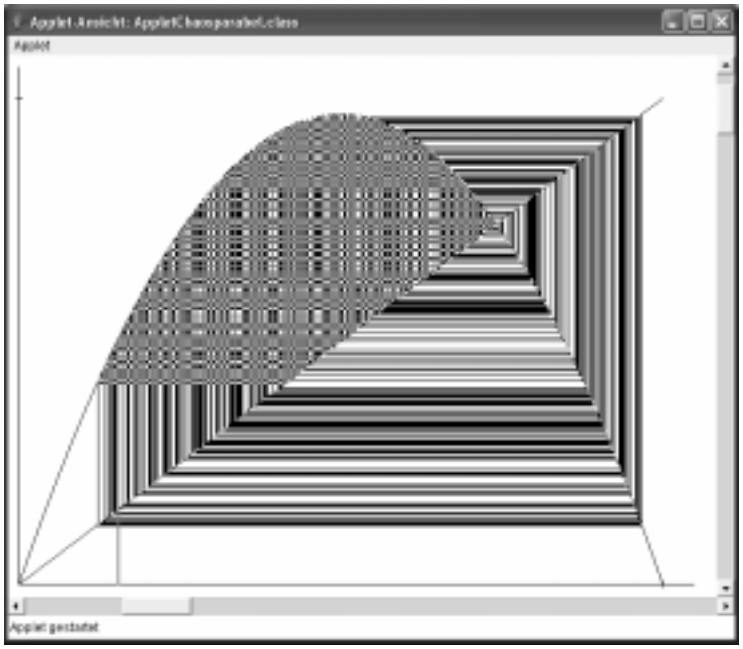
$$x_{t+1} = a \cdot x_t \cdot (1 - x_t)$$

Zeichnen wir den zugehörigen Funktionsgraphen, dann erhalten wir eine nach unten geöffnete Parabel, die die x-Achse in den Punkten 0 und 1 schneidet. Der Parameter a bestimmt die Steilheit der Parabel und kann durch eine Scrollbar verändert werden, die am rechten Rand des Fensters klebt. Den Startwert x_1 können wir ebenfalls mit Hilfe einer Scrollbar-Komponente bestimmen. Der erste berechnete Funktionswert wird dann durch eine senkrechte Linie von diesem Wert zur Parabel angezeigt.

Es fehlt jetzt nur noch eine grafische Veranschaulichung des Wieder-Hineinsteckens des Funktionswerts in die Berechnungsvorschrift. Das Vertauschen der Rollen von x- und y-Wert entspricht in der Grafik einer Vertauschung der beiden Koordinatenachsen, und das kann durch eine Spiegelung an der Geraden $y=x$, also der Hauptdiagonalen geschehen. Wir zeichnen deshalb eine horizontale Linie vom erreichten Punkt der Parabel zu dieser Diagonalen, erhalten damit den neuen x-Wert und bestimmen den zugehörigen Funktionswert, indem wir wiederum senkrecht nach oben (oder unten) zur Parabel zeichnen. Dieser Prozess wird dann laufend wiederholt. Das Ergebnis ist eine Art Treppenkurve, die unabhängig vom Startwert immer am Schnittpunkt von Parabel und Diagonale endet – wenn denn der Parameter a nicht zu groß und damit die Parabel nicht zu steil wird.

[15] Ein Beispiel für die Herleitung einer Beziehung dieser Art folgt im nächsten Kapitel.

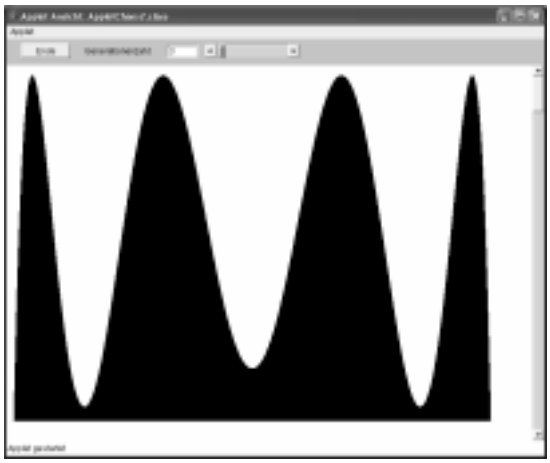




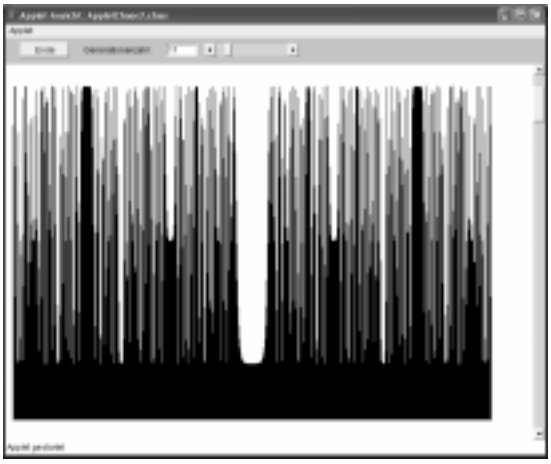
Vergrößern wir jetzt den Wert von *a*, dann verläuft die Treppenkurve (oft spiralförmig) immer noch auf diesen Schnittpunkt zu, benötigt aber deutlich mehr Rechenschritte bis zum Ende. Bei weiterer Erhöhung läuft die Kurve in periodische Zustände hinein, bei denen sie ohne weitere Annäherung an den Schnittpunkt immer wieder den gleichen Kurvenzug beschreibt. Bei noch größeren Werten von *a* (und noch steilerer Parabel) erreicht die Treppenkurve den Schnittpunkt nicht mehr. Stattdessen springt sie völlig unregelmäßig mal nahe an den Schnittpunkt heran und dann wieder weit von ihm weg. Leichte Veränderungen des Anfangswertes führen zu völlig anderen Verläufen der Iteration. Der Kurvenverlauf wird *chaotisch*.

Was bedeutet das nun?

Interpretieren wir den Startwert x_1 als „das Wetter von heute“, dann bedeutet das Ergebnis des 30. Iterationsschritts „das Wetter in einem Monat“. Dieses Ergebnis ändert sich aber schlagartig (praktisch im gesamten möglichen Wertebereich), wenn man den Startwert nur minimal ändert. Bestehen nur geringste Unsicherheiten z. B. über die Temperaturverteilung heute (und die bestehen immer), dann ist das Wetter schon über eine geringe Zeitspanne praktisch unvorhersagbar! Anschaulich spricht man davon, dass „ein Schmetterlings-Flügelschlag in Südamerika nach einiger Zeit einen Taifun in China“ verursachen kann.

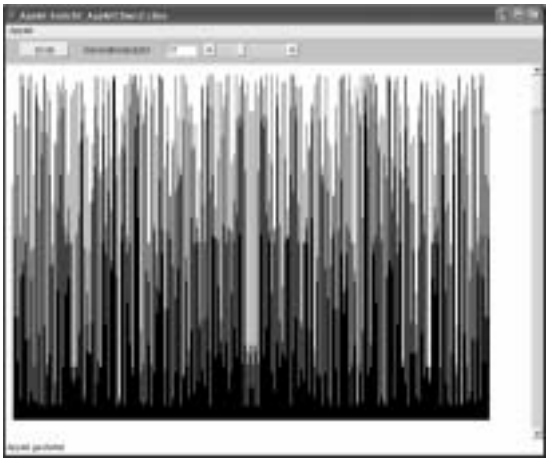


Den Zusammenhang zwischen der Änderung des Startwertes und dem Ergebnis nach einer festen Zahl von Iterationsschritten lässt sich grafisch leicht darstellen, wenn man über dem Startwert den nach n Iterationen berechneten Endwert aufträgt.



Iteriert man nur dreimal („Dreitagewetter“), dann ergibt sich eine „glatte“ Kurve: benachbarte Startwerte führen zu ähnlichen Ergebnissen.

Bei n=11 („11-Tage-Wettervorhersage“) ergibt sich zwar in der Mitte des Intervalls noch eine recht glatte Kurve, an den Seiten springen die Ergebnisse aber schon sehr willkürlich.



Für n=31 („die Monats-Wettervorhersage“) springen die Ergebnisse praktisch beliebig zwischen benachbarten Werten. Eine Vorhersage des Ergebnisses ist unmöglich.

3.10.2 Populationsentwicklungen

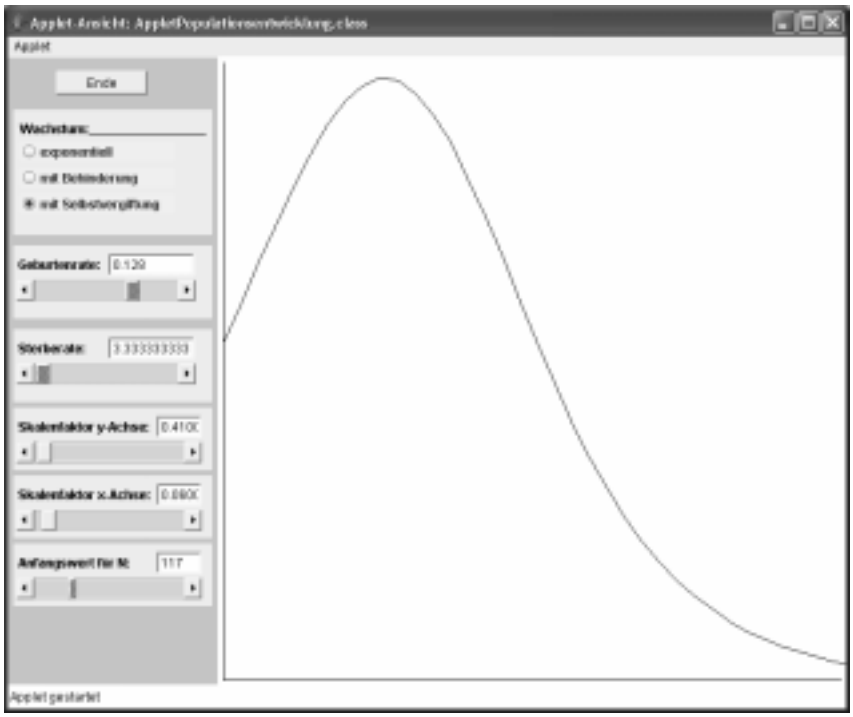
Ein weiterer Anwendungsfall für Iterationen ist die Simulation der Bevölkerungsentwicklung in einem abgeschlossenen Lebensraum, in dem nur gleichartige Individuen leben. (Als Beispiel kann eine Bakterienkultur in einer Petrischale dienen.). Gilt für diesen Raum eine *Geburtenrate* g und eine *Sterberate* s der Individuen, dann erhält man aus einfachen Überlegungen die Bevölkerungszahl N_{t+1} einer Generation aus der vorhergehenden N_t zu

$$N_{t+1} = N_t + (g - s) \cdot N_t$$

Ist die Sterberate konstant und größer als die (konstante) Geburtenrate, dann stirbt die Population aus, ist sie kleiner, dann wächst die Bevölkerung exponentiell. (***Modell 1: exponentielles Wachstum***)

Man kann die Simulation aber noch verbessern, indem man die Sterberate von der Bevölkerungszahl abhängig macht. Ist die Sterberate proportional zu N (z. B. weil sich die Ernährungslage bei wachsender Bevölkerungszahl verschlechtert), dann haben wir das ***Modell 2: Wachstum mit Behinderung***.

Ist die Sterberate proportional zur Gesamtzahl der Individuen, die bisher im Lebensraum gelebt haben (z. B. weil diese die Umwelt vergiften oder Ressourcen verbraucht haben), dann spricht man vom ***Modell 3: Wachstum mit Selbstvergiftung***.



Wir können das ausgewählte Modell durch einen Satz von *Radiobuttons* bestimmen, und die Geburten- und Sterberate mit *Schiebereglern* einstellen. Veränderte Skalenfaktoren für die Achsen und eine Verschiebung des Anfangswerts gestatten es dann, beliebige Ausschnitte der gezeichneten Kurve zu betrachten.

Überlassen wir das Bestimmen der Anfangswerte und das Zeichnen des Koordinatensystems der *paint*-Methode, dann ändert sich beim Drücken z. B. des Radiobuttons *rbExponentiell*, mit dem das Modell Nr.1 ausgewählt wird, nur die entsprechende Ganzzahlvariable *modell*. Zusätzlich wird ein geeigneter Teiler zur Anpassung der Sterberate gewählt. Danach geht das Zeichnen neu los.

```
public void actionPerformed(java.awt.event.ActionEvent e)
{
  modell = 1;
  teiler = 1000;
  repaint();
}
```

... und Neuanfang

Alle für das Zeichnen erforderlichen Aktionen laufen in der *paint*-Methode ab. Zuerst werden aus den Positionen der Schieberegler die erforderlichen Größen bestimmt. Danach wird gezeichnet, solange sich der Graph im Zeichenbereich befindet. Da bei exponentiellem Wachstum Zahlenüberläufe möglich sind, werden die Funktionswerte wieder in einem *try...catch...*-Block berechnet.

```
public void paint(Graphics g)
{
  double gebRate = 2.0*sbGebRate.getValue()/1000;
  tfGebRate.setText(""+gebRate);

  double sterbRate = 1.0*sbSterbRate.getValue()/teiler;
  tfSterbRate.setText(""+sterbRate);

  double dy = 0.01+1.0*sbYskala.getValue()/5;
  tfYskala.setText(""+dy);

  double dx = 0.01+1.0*sbXskala.getValue()/100;
  tfXskala.setText(""+dx);

  int anfN = sbAnfN.getValue();
  tfAnfN.setText(""+anfN);

  int x0 = 205;
  int y0 = getHeight()-5;
  int xMax = getWidth()-5;

  long n = anfN;
  long nGesamt = n;
```

Erforderliche Größen aus den Positionen der Schieberegler ermitteln.

Koordinaten des Ursprungs und Breite der x-Achse

Anfangswerte der Population

```
  int xAlt=x0,yAlt=anfN;
  int xNeu,yNeu;

  int generation = 0;

  g.setColor(Color.BLACK);
  g.drawLine(x0,5,x0,y0);
  g.drawLine(x0,y0,xMax,y0);

  do
  {
    try
    {
      if(modell==1)
        n = Math.round(n + (gebRate-sterbRate)*n);
      else
        if(modell==2)
          n = Math.round(n + (gebRate-sterbRate*n)*n);
        else
          if(modell==3)
            n = Math.round(n + (gebRate-sterbRate*nGesamt)*n);
      if(n<0) n=0;
    }
    catch(Exception e){n = 0;}
    nGesamt = nGesamt + n;

    xNeu = (int)Math.round(generation/dx+x0);
    yNeu = (int)Math.round(y0-n/dy);
    if(yNeu > 5) g.drawLine(xAlt,yAlt,xNeu,yNeu);

    xAlt = xNeu;
    yAlt = yNeu;
    generation++;
  }
  while((xNeu<=xMax) && (yNeu > 5));
}
```

Endpunkte der zu zeichnenden Teilstrecke

Koordinatensystem

je nach Modell Populationszahl berechnen

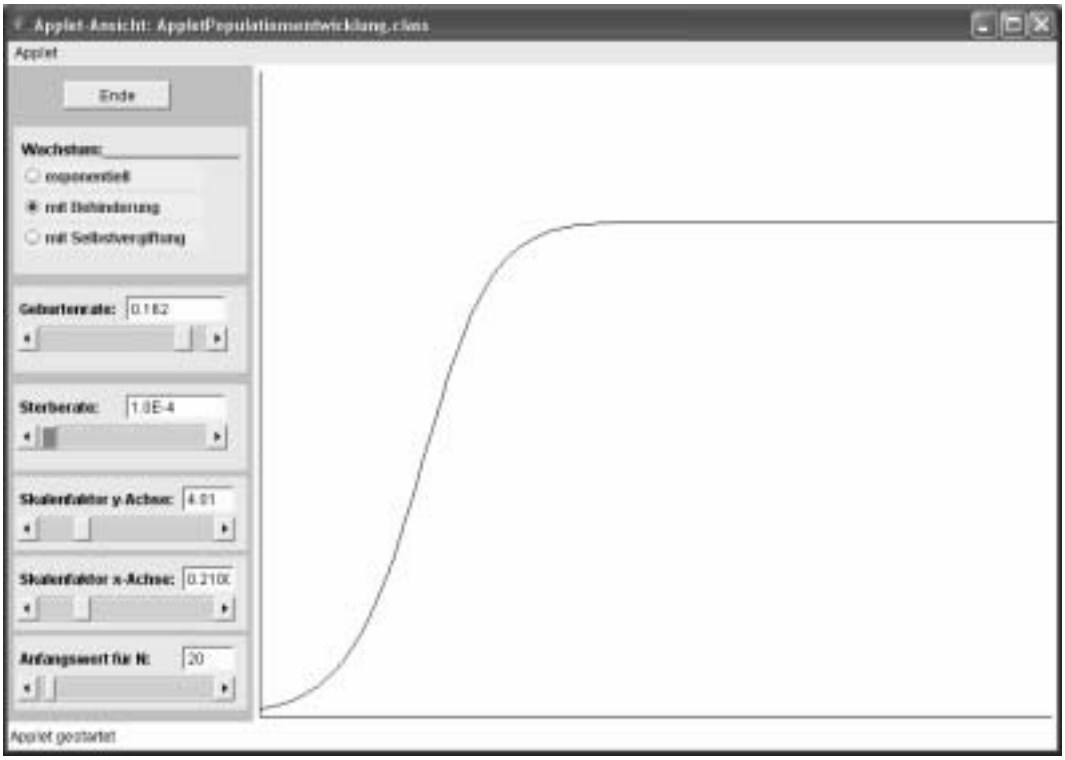
sicherheitshalber

nur sinnvolle Werte berechnen

für Modell 3

Punkte berechnen und ggf. zeichnen

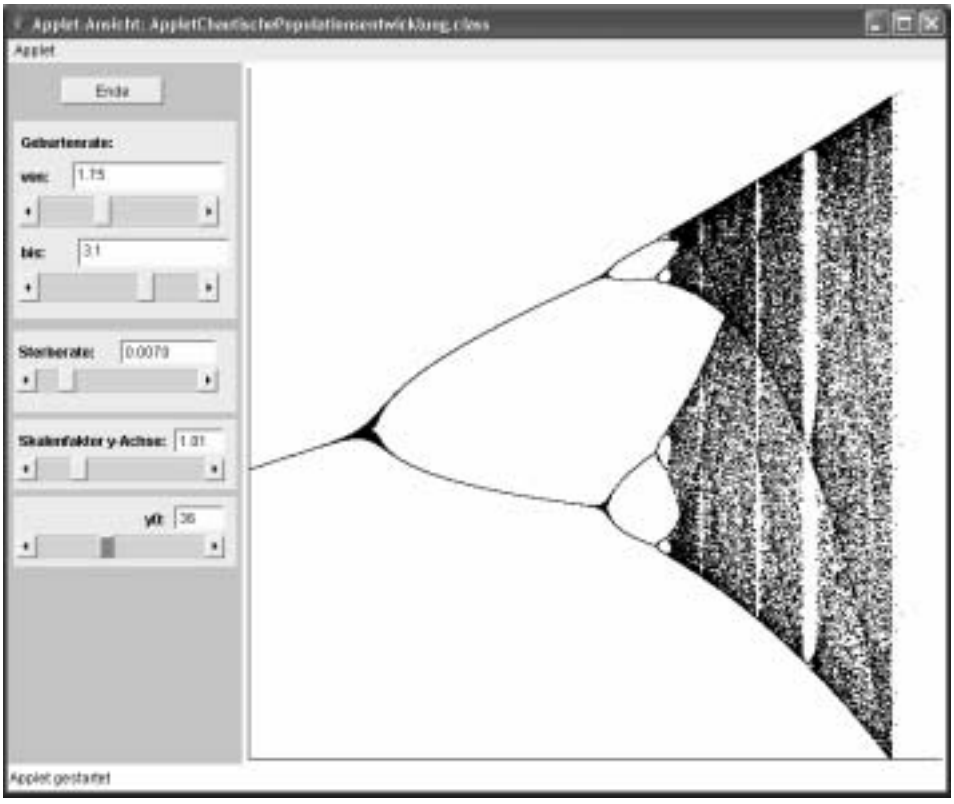
Interessant ist das Modell 2 („Wachstum mit Behinderung“), bei dem sich nach einigen Generationen ein Gleichgewichtszustand einstellt, der sich in einer konstanten Bevölkerungszahl bemerkbar macht.



3.10.3 Chaotische Bevölkerungsdynamik

Wir wollen uns jetzt für die Frage interessieren, wie die stabile Bevölkerung nach Modell 2, also die nach einigen Generationen erreichte Bevölkerungszahl, bei fester Sterberate von der Geburtenrate abhängt. Dazu variieren wir die Geburtenrate in einem vorgegebenen Bereich, berechnen mit immer gleichem Startwert einige Iterationswerte (z. B. 50 Generationen) und tragen die nächsten (z. B. 200) Generationen im Diagramm über der Geburtenrate auf. Wir erwarten natürlich, dass sich (zumindest fast) immer der gleiche Wert ergibt, so dass diese Werte alle dicht um oder auf einem Punkt liegen.

Anfangs erhalten wir auch so ein Ergebnis, und da die stabile Bevölkerungszahl mit steigender Geburtenrate zunimmt, ergibt sich eine ansteigende Gerade. Aber dann passiert etwas! Mit weiter steigender Geburtenrate „springt“ die Bevölkerungszahl plötzlich zwischen zwei Werten, und diese Äste gabeln später wieder und wieder. Man spricht von *Bifurkationsprozessen*. Im rechten Teil des Diagramms schwankt die Bevölkerungszahl von Generation zu Generation fast beliebig, und nahe beieinander liegende Geburtenraten führen zu unvorhersagbaren Sprüngen im Ergebnis. Das Verhalten wird chaotisch.



3.10.4 Aufgaben

1. Schreiben Sie ein Programm, das

   a: die **Chaosparabel** entsprechend den Bildern in Kapitel 3.9.1 erzeugt.

   b: die Ergebnisse einer einzustellenden Zahl von Iterationen der **logistischen Gleichung** wie in Kapitel 3.9.1 angegeben darstellt.

   c: die **Entwicklung der Populationen** nach den drei Modellen wie in Kapitel 3.9.2 angegeben darstellt.

   d: das **Bifurkationsdiagramm** aus Kapitel 3.9.3 erzeugt. Die Skalierung der y-Achse und das Intervall für die Geburtenrate sollen veränderbar sein. Benutzen Sie als Variable für die Anzahl der Individuen eine Gleitpunktzahl (z. B. *double*) und runden Sie erst bei der grafischen Darstellung.

   e: das Bifurkationsdiagramm aus Kapitel 3.9.3 erzeugt. Mit Hilfe der Maus sollen **rechteckige Ausschnitte** aus dem Diagramm markiert werden können. Diese werden dann in einem nächsten Schritt vergrößert dargestellt.

2. Leben Hasen und Füchse in einem gemeinsamen Lebensraum, dann haben wir es mit einem ***Räuber-Beute-System*** zu tun: Füchse fressen Hasen und vermehren sich dabei gut. Leider nimmt dadurch aber die Hasenzahl ab und die nächste Fuchsgeneration muss hungern, wobei viele sterben. Dadurch geht es späteren Hasen besser usw. Die neue Hasenzahl H_{n+1} ergibt sich aus der alten H_n, vermehrt um die Neugeborenen, deren Zahl proportional zu H_n ist (c_1*H_n), und vermindert um die Zahl der Gefressenen, deren Zahl von der Anzahl der Begegnungen zwischen Hasen und Füchsen abhängt ($c_2*H_n*F_n$): $H_{n+1} = H_n + c_1 \cdot H_n - c_2 \cdot H_n \cdot F_n$

   Ebenso erhält man die neue Fuchsanzahl F_{n+1} aus der alten F_n, vermindert um die Zahl der gestorbenen und vermehrt um die Zahl der neu geborenen, deren Zahl vom Futterzustand, also der Zahl der Begegnungen zwischen Hasen und Füchsen abhängt.
   $$F_{n+1} = F_n - c_3 \cdot F_n - c_4 \cdot H_n \cdot F_n$$

   Wenn wir die Zahl der Hasen auf der y-Achse und die Zahl der Füchse auf der x-Achse auftragen, dann erhalten wir Punkte in der Hasen-Füchse-Ebene - im *Phasenraum*.

   a: Stellen Sie die Entwicklung des Systems als Punktfolge in einem Diagramm dar.

   b: Experimentieren Sie mit den Konstanten des Systems und ermitteln Sie deren Einfluss. Welche Auswirkungen werden durch das Modell bestimmt, welche durch Einflüsse der Computerarithmetik. Wie kann man das feststellen?

   c: Führen Sie Möglichkeiten ein, die Konstanten am Bildschirm zu verändern.

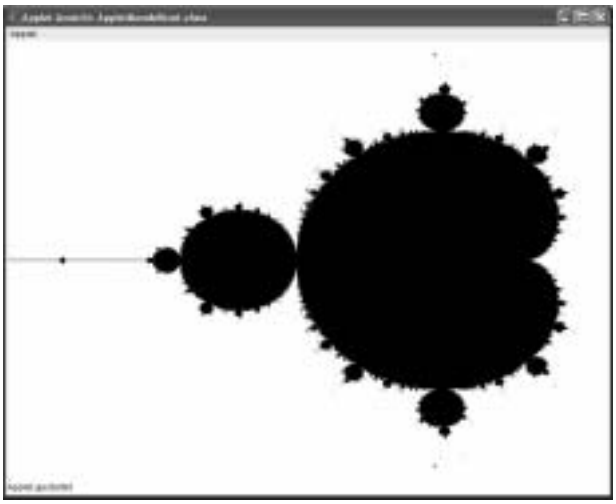
3. Eine berühmte **Iterationsvorschrift** startet mit den Koordinaten eines Punktes C(a|b) in der Ebene. Aus diesen werden die Koordinaten der Folgepunkte $X_n(x_n|y_n)$ nach der folgenden Vorschrift bestimmt:

   im ersten Schritt: $x_1 = a;$ $y_1 = b;$

   weitere Schritte: $x_{n+1} = x_n^2 - y_n^2 + a;$ $y_{n+1} = 2*x_n*y_n + b;$

   Da es sich um eine Art Quadrierung mit nachfolgender Addition handelt, sollten Punkte in der Nähe des Ursprungs zu einer Punktfolge führen, die gegen den Ursprung konvergiert, während Punktfolgen, die in größerer Entfernung vom Ursprung starten, ins Unendliche laufen. Wir wollen jetzt die Punkte einer Umgebung des Ursprungs (z. B. innerhalb des Rechtecks mit den Eckpunkten (-2.0|1.1) und (0.6|-1.1)) als Startwerte auffassen und dann jeweils die nächsten 100 Punkte der angegebenen Punktfolge berechnen. Liegt der letzte Punkt nahe beim Ursprung (z. B. näher als 2 Längeneinheiten), dann wollen wir den ***Startpunkt*** schwarz färben, sonst weiß. Die Rechnungen sind sehr zeitaufwändig und sollten deshalb direkt in zwei geschachtelten Schleifen ausgeführt werden, in denen die a- bzw. die b-Koordinate des Startwerts in den angegebenen Grenzen verändert wird.

```
FÜR i VON 0 BIS RechtenBildrand TUE
    FÜR j VON 0 BIS UnterenBildrand TUE
        Berechne aus i und j die Koordinaten des Startpunktes
        Führe das Iterationsverfahren wie angegeben durch
        Färbe den Punkt (i|j) je nach Ergebnis des Iterationsprozesses ein
```



Das Ergebnis ist die bekannte ***Mandelbrotmenge***, das ***„Apfelmännchen“***.

Nehmen Sie an, dass alle Operationen im Rechner gleich lange dauern. Schätzen Sie dann die Rechenzeit (die „*Zeitkomplexität*“) in Abhängigkeit von den Appletabmessungen ab.