Prof. Andreas Goerdt TU Chemnitz
M. Pippig, M. Riedel, U. Schönke Wintersemester 2003/2004

Algorithmen und Programmierung

11. Übung – Lösungsvorschläge

1. Aufgabe:

```
public static double p(int n, double x) {

  double a = IOTools.readDouble("Naechster␣Koeffizient:␣");

  if (n==0)
    return a;
  else
    return (x*p(n-1,x)+a);
}
```

2. Aufgabe:

$$\begin{aligned} f(0,0,1) &= 0 \\ f(1,0,1) &= 1 \\ f(n,0,1) &= f(n-1,0,1)+f(n-2,0,1) \quad \text{für } n \geq 2 \end{aligned}$$

Der Aufruf von `f(n,0,1)` liefert die *n*-te Fibonaccizahl zurück.

3. Aufgabe:

```
public static boolean palindrom(int n) {
  boolean ok;
  char a,b;

  if (n<2){
    // Mitte ist erreicht
    // gerade Anzahl -> nichts einlesen
    // ungerade Anzahl -> Buchstabe einlesen,
    // aber nicht vergleichen
    if (n==1) {
      a = IOTools.readChar("Naechster␣Buchstabe:␣");
    }
    return true;
  }
```

```
    else {
      a = IOTools.readChar("Naechster␣Buchstabe:␣");
      ok = palindrom(n-2);
      b = IOTools.readChar("Naechster␣Buchstabe:␣");
      return ((a==b)&&ok);
    }
  }
```

4. Aufgabe:

   Herangehensweise: Es müssen alle Wege „durchprobiert“ werden. Dafür berechnen wir für jedes Feld, ob es sinnvoller/besser ist, nach rechts oder nach unten zu gehen. Das können wir natürlich nur entscheiden, indem wir **alle** Wege über rechts und **alle** Wege über unten betrachten.

```
public static int max_gold(int[][] spielfeld,
                           int zeile, int spalte) {

  int unten, rechts;

  // Abbruch auf Feld spielfeld[m][n]
  if ((zeile==spielfeld.length-1)&&
      (spalte==spielfeld[0].length-1)){
    return spielfeld[zeile][spalte];
  }
  else {
    // noch nicht fertig
    // unterer Rand erreicht?
    if (zeile==spielfeld.length-1) {
      // weiter nach rechts
      return spielfeld[zeile][spalte] +
             max_gold(spielfeld,zeile,spalte+1);
    }
    else {
      // rechter Rand erreicht?
      if(spalte==spielfeld[0].length-1) {
        // weiter nach unten
        return spielfeld[zeile][spalte] +
               max_gold(spielfeld,zeile+1,spalte);
      }
      else {
        // noch irgendwo mitten im Feld
        unten = max_gold(spielfeld,zeile+1,spalte);
        rechts = max_gold(spielfeld,zeile,spalte+1);
```

```
            if (unten>rechts) {
              return spielfeld[zeile][spalte] + unten;
            }
            else {
              return spielfeld[zeile][spalte] + rechts;
            }
          }
        }
      }
    }
```

Hinweis: Es geht auch kürzer!

5. Aufgabe:

   a) Die Kurve $S(i)$ ist folgendermaßen aufgebaut:

$$S(i) : A(i) \searrow B(i) \swarrow C(i) \nwarrow D(i) \nearrow$$

   Das Rekursionsmuster ergibt sich wie folgt:

$$\begin{array}{lllllllll} A(i): & A(i-1) & \searrow & B(i-1) & \rightarrow & D(i-1) & \nearrow & A(i-1) \\ B(i): & B(i-1) & \swarrow & C(i-1) & \downarrow & A(i-1) & \searrow & B(i-1) \\ C(i): & C(i-1) & \nwarrow & D(i-1) & \leftarrow & B(i-1) & \swarrow & C(i-1) \\ D(i): & D(i-1) & \nearrow & A(i-1) & \uparrow & C(i-1) & \nwarrow & D(i-1) \end{array}$$

   Dabei bedeuten diagonale Pfeile ($\searrow, \swarrow, \nearrow, \nwarrow$) Linien in der entsprechenden Richtung mit der Länge einer Diagonale durch ein Quadrat der Einheitslänge und horizontale bzw. vertikale Pfeile ($\rightarrow, \downarrow, \leftarrow, \uparrow$) Linien in der entsprechenden Richtung mit doppelter Einheitslänge.

b)

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

public class SierpinskiKurve extends Applet {

  public static void suedost(Graphics g, Point a, int k){

    int xneu, yneu;
    xneu=a.x+k;
    yneu=a.y+k;
    g.drawLine(a.x,a.y,xneu,yneu);
    a.x=xneu;
    a.y=yneu;
  }

  public static void suedwest(Graphics g, Point a, int k){

    int xneu, yneu;
    xneu=a.x-k;
    yneu=a.y+k;
    g.drawLine(a.x,a.y,xneu,yneu);
    a.x=xneu;
    a.y=yneu;
  }

  public static void nordwest(Graphics g, Point a, int k){

    int xneu, yneu;
    xneu=a.x-k;
    yneu=a.y-k;
    g.drawLine(a.x,a.y,xneu,yneu);
    a.x=xneu;
    a.y=yneu;
  }

  public static void nordost(Graphics g, Point a, int k){

    int xneu, yneu;
    xneu=a.x+k;
    yneu=a.y-k;
    g.drawLine(a.x,a.y,xneu,yneu);
    a.x=xneu;
    a.y=yneu;
  }
```

```
public static void A(Graphics g, Point a, int i, int k) {

  int xneu;

  if (i>0) {
    A(g,a,i-1,k);
    suedost(g,a,k);
    B(g,a,i-1,k);

    xneu=a.x+2*k;
    g.drawLine(a.x,a.y,xneu,a.y);
    a.x=xneu;

    D(g,a,i-1,k);
    nordost(g,a,k);
    A(g,a,i-1,k);
  }
}

public static void B(Graphics g, Point a, int i, int k) {
  int yneu;

  if (i>0) {
    B(g,a,i-1,k);
    suedwest(g,a,k);
    C(g,a,i-1,k);

    yneu=a.y+2*k;
    g.drawLine(a.x,a.y,a.x,yneu);
    a.y=yneu;

    A(g,a,i-1,k);
    suedost(g,a,k);
    B(g,a,i-1,k);
  }
}

public static void C(Graphics g, Point a, int i, int k) {
  int xneu;

  if (i>0) {
    C(g,a,i-1,k);
    nordwest(g,a,k);
    D(g,a,i-1,k);
```

```
        xneu=a.x-2*k;
        g.drawLine(a.x,a.y,xneu,a.y);
        a.x=xneu;

        B(g,a,i-1,k);
        suedwest(g,a,k);
        C(g,a,i-1,k);
      }
    }

    public static void D(Graphics g, Point a, int i, int k) {
       int yneu;

      if (i>0) {
        D(g,a,i-1,k);
        nordost(g,a,k);
        A(g,a,i-1,k);

        yneu=a.y-2*k;
        g.drawLine(a.x,a.y,a.x,yneu);
        a.y=yneu;

        C(g,a,i-1,k);
        nordwest(g,a,k);
        D(g,a,i-1,k);
      }
    }

    public void paint(Graphics g) {
      Point a = new Point(50, 50); //Startpunkt
      int tiefe=3;
      int laenge=32/tiefe;

      A(g,a,tiefe,laenge);
      suedost(g,a,laenge);
      B(g,a,tiefe,laenge);
      suedwest(g,a,laenge);
      C(g,a,tiefe,laenge);
      nordwest(g,a,laenge);
      D(g,a,tiefe,laenge);
      nordost(g,a,laenge);
    }
}
```