TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Praktikum Grundlagen der Programmierung
F. Forster, T. Gawlitza, A. Flexeder

WS 2007/2008
Lösungsvorschläge zu Blatt 4
9. November 2007

Übungen zu Praktikum Grundlagen der Programmierung

Aufgabe 22 Fibonacci-Folge (Lösungsvorschlag)

```
public class Fib extends MiniJava {
    public static int fib_recursive(int i) {
        int fib;
        if (i == 0)
            fib = 0;
        else if (i == 1)
            fib = 1;
        else
            fib = fib_recursive(i - 1) + fib_recursive(i - 2);
        return fib;
    }

    public static int fib_iterative(int i) {
        int fib = 0;

        if (i == 0)
            fib = 0;
        else if (i == 1)
            fib = 1;
        else {
            // fib(i) = fib(i-1) + fib(i-2);
            int fib_minus_zwei = 0;
            int fib_minus_eins = 1;
            for(int count = 2; count <= i; count++) {
                // berechne fib(count)
                fib = fib_minus_zwei + fib_minus_eins;

                // sichere Werte fuer naechste Iteration
                fib_minus_zwei = fib_minus_eins;
                fib_minus_eins = fib;
            }
        }
        return fib;
    }

    public static void main(String[] args) {
        // Eingabe
        int i = read();
        if (i < 0) {
            write("Nur␣positive␣Eingaben␣erlaubt.");
        } else {
            write("Iterativ:␣fib(" + i + ")␣=␣" + fib_iterative(i));
            write("Rekursiv:␣fib(" + i + ")␣=␣" + fib_recursive(i));
        }
    }
}
```

Aufgabe 23 Labyrinth (Lösungsvorschlag)

```
public class MazeSolution extends Maze{
    static boolean[][] maze;
    static int goalX, goalY;

    public static void main(String[] args) {
        int width = 10;
        int height = 10;
        goalX = width - 1;
        goalY = height - 2;
        maze = generateMaze(width, height);
        walk(1, 0, 1);
  //alternativ: walk2(1,0,1);
    }

    public static void walk(int x, int y, int direction) {
        draw(x, y, maze);
        if (x == goalX && y == goalY)
            return;

        if (x == 1 && y == 0 && direction != 1) {
            System.out.println("There␣is␣no␣way␣out␣:(");
            return;
        }
        // do we have a wall on the right hand side?
        if (direction == 0 && maze[x - 1][y] || direction == 1
            && maze[x][y + 1] || direction == 2 && maze[x + 1][y]
            || direction == 3 && maze[x][y - 1]) {

            // is there an obstacle directly in front of us?
            if (direction == 0 && maze[x][y + 1] || direction == 1
                && maze[x + 1][y] || direction == 2 && maze[x][y - 1]
                || direction == 3 && maze[x - 1][y]) {
                System.out.println("There␣is␣an␣obstacle,␣turning␣to␣"
                                   + ((direction + 1) % 4));
                // we can turn counterclockwise, having the obstacle on our
                // right side
                walk2(x, y, ((direction + 1) % 4));

            } else {
                // we walk straigt on
                if (direction == 0)
                    walk2(x, y + 1, direction);
                else if (direction == 1)
                    walk2(x + 1, y, direction);
                else if (direction == 2)
                    walk2(x, y - 1, direction);
                else
                    walk2(x - 1, y, direction);
            }
        } else {
            // there is no wall on the right side => we walk to the right side
            // and turn until we have a wall on the right side
            if (direction == 0)
                walk2(x - 1, y, (direction + 3) % 4);
            else if (direction == 1)
                walk2(x, y + 1, (direction + 3) % 4);
            else if (direction == 2)
                walk2(x + 1, y, (direction + 3) % 4);
            else
                walk2(x, y - 1, (direction + 3) % 4);
        }
    }


    /** ********************Alternativloesung************************************** */
    public static int getXOffsetRechts(int direction) {
        int ret = (direction + 1) % 2;
        if (direction / 2 == 0) return -ret;
        return ret;
    }
    public static int getYOffsetRechts(int direction) {
        return -getXOffsetRechts((direction+3)%4);
    }
    public static int getXOffsetDavor(int direction) {
        return -getXOffsetRechts((direction+3)%4);
```

```
    }
    public static int getYOffsetDavor(int direction) {
        return -getXOffsetRechts(direction);
    }

    public static void walk2(int x, int y, int direction) {
        draw(x, y, maze);
        if (x == goalX && y == goalY)
            return;
        if (x == 1 && y == 0 && direction != 1) {
            System.out.println("There␣is␣no␣way␣out␣:(");
            return;
        }
        if (maze[x + getXOffsetRechts(direction)][y + getYOffsetRechts(direction)]) {
            // a wall to the right
            if (maze[x + getXOffsetDavor(direction)][y + getYOffsetDavor(direction)]) {
                // a wall in front
                System.out.println("There␣is␣an␣obstacle,␣turning␣to␣" + ((direction + 1) % 4));
                // we can turn counterclockwise, having the wall on our right side
                walk(x, y, ((direction + 1) % 4));
            } else // walk straight forward
                walk(x + getXOffsetDavor(direction), y + getYOffsetDavor(direction), direction);
        } else {
            // there is no wall on the right side => we walk to the right side
            // and turn until we have a wall on the right side
            walk(x + getXOffsetRechts(direction), y + getYOffsetRechts(direction), (direction + 3) % 4);
        }
    }
}
```

Aufgabe 24 (Ü) Quadrat-Fraktale

```
public class SquareFractalSolution extends SquareFractal{
    public static void main( String[] args ) {
  int iterations = Integer.parseInt(args[0]);
  initFrame();
  squareFractal(200,200,100,iterations,0);
    }

    public static void squareFractal(int x, int y, int length, int iteration, int orientation){
  if (iteration <= 0) // exit criteria
      return;

  // draw this iteration
  drawRectangle((x-length/2),(y-length/2),length,length);

  // draw next iterations, depending on orientation
  orientation = orientation % 4;
  if (orientation != 3)
      squareFractal((x-2*length/3), (y), (length/3), iteration-1,1);
  if (orientation != 0)
      squareFractal((x), (y+2*length/3), (length/3), iteration-1,2);
  if (orientation != 1)
      squareFractal((x+2*length/3), (y), (length/3), iteration-1,3);
  if (orientation != 2)
      squareFractal((x), (y-2*length/3), (length/3), iteration-1,0);
    }
}
```