Allgemeines Programmierpraktikum **Sommersemester 2018**
Dr. Henrik Brosenne Georg-August-Universität Göttingen
Institut für Informatik

Übung 5

Abgabe der Prüfsumme bis Mi., 06.06., 23:59 Uhr.
Testat von Do., 07.06., 12-14 Uhr bis Di., 12.06., 16-18 Uhr.

Hilfe zum Bearbeiten der Übungen können Sie grundsätzlich in jeder Rechnerübungen bekommen, insbesondere in den Rechnerübungen **Mi., 06.06.** in denen **keine Testate** stattfinden.

Vorankündigung

Projekt

Im Anschluss an diese Übung beginnt das Projekt.

Vorbereitung

Java

Informieren Sie sich in der *Java Platform, Standard Edition 8 API Specification* `https://docs.oracle.com/javase/8/docs/api/` über die Swing-Komponenten `javax.swing.JButton` und `javax.swing.Box`, sowie die Layout-Manager `java.awt.BorderLayout` (default bei `JFrame`) und `java.awt.FlowLayout` (default bei `JPanel`). Informieren Sie sich weiterhin über `javax.imageio.ImageIO` und `java.awt.image.BufferedImage`.

Aufgabe 1 – 50 Punkte

Grafik

Programmieren Sie ein Anwendung, die auf der Kommandozeile den Namen eine Grafikdatei übergeben bekommt und ein Fenster öffnet, um die Datei darzustellen.

Im oberen Teil des Fensters befinden sich drei Schaltflächen *Original*, *Grayscale*, *Pattern*. In der Mitte befindet sich ein Bereich, in dem die Grafik angezeigt wird, deren Darstellung über die Schaltflächen verändert werden kann. **Unten rechts** ist eine Schaltfläche um die Anwendung zu beenden.

Das Fenster könnte wie folgt aussehen mit der in der Stud.IP-Veranstaltung unter *Dateien→Übung* hinterlegten Grafik `kandinsky.jpg`.

Die Anzeigefläche für die Grafik ist so groß, dass ein Pixel aus der Grafikdatei durch ein Rechteck der Größe 2x2 Pixel dargestellt werden kann.

Button *Original*/Standard

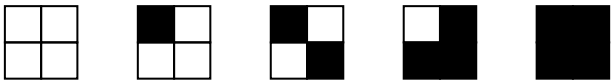
Die Farbe (RGB/rot-grün-blau Wert) jedes Pixel der Grafik wird gelesen und ein Rechteck in dieser Farbe gezeichnet.

Button *Grayscale*

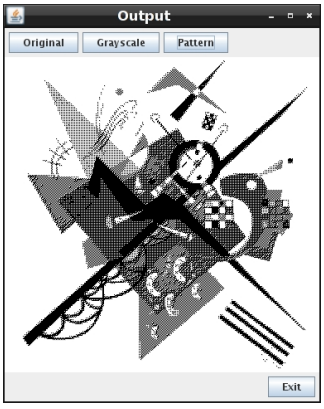
Die Farbe (RGB-/Rot-Grün-Blau-Wert) jedes Pixel der Grafik wird gelesen. Es wird jeweils der Rot-, Grün und Blau-Wert ermittelt und deren Mittelwert berechnet, dann wird ein Rechteck gezeichnet, wobei der Rot-, Grün- und Blau-Wert der Zeichenfarbe der berechnete Mittelwert ist.

Button *Pattern*

Der Mittelwert für jeden Pixel wird wie bei *Grayscale* berechnet. Abhängig davon welche RGB-Werte verwendet werden ergibt sich ein Wertebereich für den Mittelwert, der in 5 Abschnitt aufgeteilt wird (z.B. kann bei Werten 0 bis 255 die Aufteilung mit Division durch 52 erfolgen). Jedem Abschnitt wird anschließend das passenden der folgenden Muster (von schwarzen und weißen Pixeln) zugeordnet und gezeichnet.



Für das obige Beispiel liefert die Pattern-Darstellung folgendes Ergebnis.



Allgemein

- Bei auftretenden Fehlern wird eine aussagekräftige Fehlermeldung geliefert.
- Verwenden Sie Ant zum automatisierten Übersetzen des Quelltext. Nach dem Übersetzen befinden sich die `java`- und die `class`-Dateien in unterschiedlichen Verzeichnissen.
- Kommentieren Sie Ihren Programmtext ausführlich. Erstellen Sie für die Klassen, Schnittstellen und alle Attribute Dokumentationskommentare für `javadoc`. Erzeugen Sie mit `ant` eine API-Dokumentation. Spärliche und/oder schlechte Kommentierung führt zu Punktabzug.

Hinweis

Als Grundlage können Sie folgendes Programm verwenden, das in der Stud.IP-Veranstaltung unter *Dateien→Übung* hinterlegt ist.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  class AppFrame extends JFrame {
6      public AppFrame(String title) {
7          super(title);
8          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9  }   }
10
11 class AppDrawPanel extends JPanel {
12     public Dimension getPreferredSize() {
13         return new Dimension(500, 200);
14     }
15
16     protected void paintComponent(Graphics g) {
17         super.paintComponent(g);
18         g.drawLine(10, 10, 490, 190);
19         g.drawString("Ein einfaches Panel", 50, 100);
20 }   }
21
22 class AppMouseAdapter extends MouseAdapter {
23     public void mouseClicked(MouseEvent e) {
24         if (e.getClickCount() > 1)
25             System.exit(0);
26 }   }
27
28 public class AppDrawEvent
29 {
30     public static void main( String[] args ) {
31         JFrame frame = new AppFrame("Allgemeines Programmierpraktikum");
32         JPanel panel = new JPanel();
33         frame.add(panel);
34
35         JPanel draw  = new AppDrawPanel();
36         JLabel label = new JLabel("Doppelklicken zum Beenden");
37         panel.add(draw);
38         panel.add(label);
39
40         AppMouseAdapter m = new AppMouseAdapter();
41         label.addMouseListener(m);
42
43         frame.pack();
44         frame.setVisible(true);
45 }   }
```

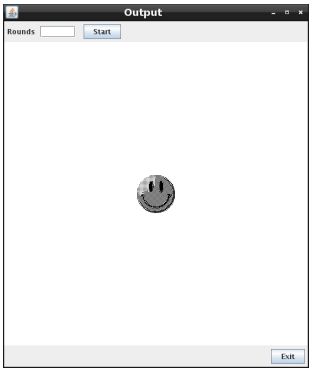
Aufgabe 2 – 50 Punkte

Grafik und Threads

Programmieren Sie ein Anwendung, die auf der Kommandozeile den Namen eine Grafikdatei übergeben bekommt und ein Fester öffnet, um einen zellulären Automaten darzustellen.

Es gibt eine Schaltfläche, um die Anwendung zu beenden. Weiterhin gibt es ein Eingabefeld, um die Anzahl der nächsten zu durchlaufenden Zeitabschnitte einzulesen und eine Schaltfläche, um die nächsten Durchläufe zu starten.

Das Fenster könnte wie folgt aussehen mit der in der Stud.IP-Veranstaltung unter *Dateien→Übung* hinterlegten Grafik `smile32.jpg`, die mit der Pattern-Darstellung aus Aufgabe 1 in 64×64 schwarz/weiß-Punkte umgewandelt wurde.

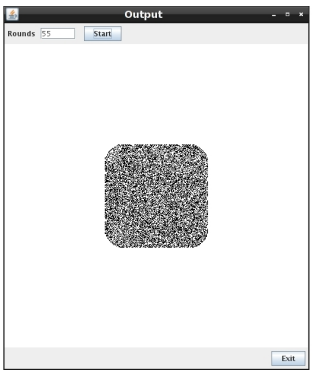


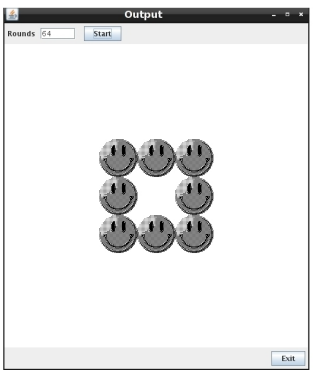
Der zelluläre Automat wird durch eine Datenstruktur dargestellt, die für jede Zelle den aktuellen und den nächsten Zustand aufnehmen kann.

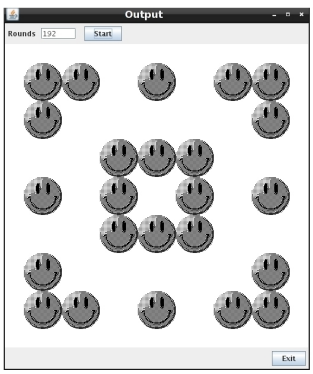
Die Datenstruktur für den zellulären Automaten wird mit toten Zellen initialisiert, lebendigen Zellen werden mit Hilfe der Grafikdatei gesetzt. Ein Möglichkeit dazu ist die Grafik in schwarz/weiß-Punkte umzurechnen und für einen schwarzen Punkte eine lebendige Zelle anzunehmen. Die Grafik wird in der Mitte des zellulären Automaten platziert und sollte nicht höher/breiter als ein viertel der Höhe/Breite des Automaten sein.

Es gibt eine Arbeiter-Thread, der die nächsten Zeitabschnitte berechnet und jeweils anzeigen lässt. Nach Erledigung seiner Aufgaben wartet (`wait`) der Arbeiter-Thread bis er benachrichtigt wird (`notify`), dass die Start-Schaltfläche betätigt wurde.

Das obige Beispiel nach 55, 64 und 192 Durchläufen. Wird eine bestimmte Anzahl von Durchläufen überschritten sind alle Zellen tot.







Allgemein

- Bei auftretenden Fehlern wird eine aussagekräftige Fehlermeldung geliefert.
- Verwenden Sie Ant zum automatisierten Übersetzen des Quelltext. Nach dem Übersetzen befinden sich die `java`- und die `class`-Dateien in unterschiedlichen Verzeichnissen.
- Kommentieren Sie Ihren Programmtext ausführlich. Erstellen Sie für die Klassen, Schnittstellen und alle Attribute Dokumentationskommentare für `javadoc`. Erzeugen Sie mit `ant` eine API-Dokumentation.

Zellulärere Automaten

Die zellulären Automaten dieser Übung sind wie folgt definiert.

- Der Automat besteht aus Zellen, die in jedem Zeitabschnitt entweder tot (0) oder lebendig (1) sind.
- Die Zellen sind in einem Quadrat angeordnet, die Seitenlänge des Quadrats ist eine Zweierpotenz.
- Jede Zelle (α) hat als Nachbarn (N_α) die jeweils angrenzenden Zellen. Für Zellen auf den Rand werden, für die fehlenden angrenzenden Zellen, die Zellen auf dem gegenüber liegenden Rand zu Nachbarn.

				N_x	N_x	N_x	
N_z	N_z		N_y	N_y	N_y		N_z
z	N_z		N_y	y	N_y		N_z
N_z	N_z		N_y	N_y	N_y		N_z
				N_x	N_x	N_x	
				N_x	x	N_x	

- Eine Zelle ist im nächsten Zeitabschnitt genau dann lebendig, wenn im aktuellen Zeitabschnitt unter ihren acht Nachbarzellen eine ungerade Anzahl lebendig ist.