Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

Animationen ohne Multithreading

Ziel: Kennenlernen elementarer Möglichkeiten einfache Animationen in Java darzustellen und erkennen, welche Probleme sich ergeben, wenn keine Multithreading verwendet wird.

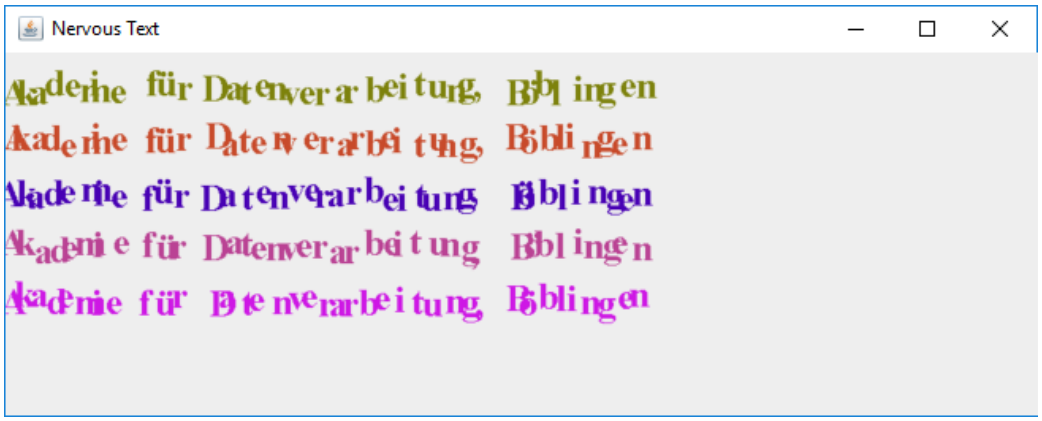
1.1. Die Klasse `TumblingDuke` soll erweitert werden, dass `ManyTumblingDukes` daraus werden:

- Bei einer Fensterbreite von 1000 Pixel und einer Höhe von 700 Pixel, sollen Reihen von „tumbling Dukes“ ihre animierten „Räder“ gleichzeitig ausführen. Die Breite des aktuellen JFrames kann man mit `this.getWidth()` ermitteln, die Höhe entsprechend mit `this.getHeight()`.
- Ein einzelnes Anmationsbild hat die Maße 130x80 (Breite x Höhe). Die Anzahl der gleichzeitigen Animationen ergibt sich also aus der Größe des JFrame und der Größe der Bilder, wobei immer nur ganze Bilder angezeigt werden sollen.
- Nach einem Durchlauf der Animation soll das „Rad“ gleich anschließend in die andere Richtung „geschlagen“ werden, dann eine kurze Verschnaufpause – und so weiter … und so weiter…

Die Grundversion von `TumblingDuke` finden Sie in Moodle, so dass diese nur entsprechend erweitert werden muss.

```
public class I13ManyTumblingDukes extends JFrame
{
    // array vom Typ Image zur Speicherung der Adressen der 17 Einzelbilder
    private Image bilder[] = new Image[17];

    public I13ManyTumblingDukes()
    {
        setBackground(Color.white);
        for (int i = 1; i <= 17; i++)
        {
            try // die Methode 'read' von 'ImageIO' verlangt eine sog. "Ausnahmebehandlung" (try-catch)
            {
                bilder[i - 1] = ImageIO.read(getClass().getResource("../TumblingDuke/T" + i + ".gif"));

            }
            catch (IOException e)
            {
                e.getMessage();     // ==> Fehlernachricht wird im Konsolenfenster angezeigt
            }
        }
    }

    public void paint(Graphics g)
    {
        int pos = 0;                    // Positionszähler für array
        int pauseNachAnimation = 2000;  // Pause von 2000 Milli-Sekunden, d.h. 2 Sek.
        int pauseNachBild = 100;        // Pause von 0,1 Sek. nach Einzelbild

        boolean bRichtung = false;  // true: vorwärts, false: rückwärts

        for (int i=0;;i++)
        {
            // Da die Koordinaten (0/0) im Rand des JFrames liegen, wird noch eine
            // zusätzliche Verschiebung um (30/10) vorgenommen
           for (int iYPos = 0; iYPos < this.getHeight() - 30 - 80; iYPos += 80)
           {
               for (int iXPos = 0; iXPos < this.getWidth() -10 - 130; iXPos += 130)
               {
                   g.drawImage(bilder[pos], iXPos + 10, iYPos + 30, this);
               }
           }

            try
            {
                if (pos == 16 || (pos == 0 && bRichtung))
                    Thread.sleep(pauseNachBild + pauseNachAnimation);
                else Thread.sleep(pauseNachBild);
            }
            catch (InterruptedException e)
            {
                System.exit(0);
            }
            if (pos == 16) bRichtung = true;
            else if (pos == 0) bRichtung = false;
            if (bRichtung) pos--;
            else pos++;
        }
    }

    public static void main(String[] args)
    {
        I13ManyTumblingDukes fenster = new I13ManyTumblingDukes();
        fenster.setSize(1000, 700);
        fenster.setVisible(true);
    }
}
```

1.2. Erstellen Sie folgende Java-Swing-Animation, die einen „nervösen“ Text ausgibt:



- Größe des JFrames: 1000/400
- Der RGB-Wert der Farbe wird zufällig bestimmt. (Siehe Methode setColor() in 28_Java_GUI_Graphics).
- Die Schrift ist "TimesRoman", fett, Schriftgröße 36 (Siehe Methode setFont() in 28_Java_GUI_Graphics).
- Jeder neue Buchstabe ist um 15 Pixel + einem zufälligen Wert von 0-10 Pixel nach rechts verschoben
- Die Zeilen erscheinen jeweils 50 Pixel untereinander, die 1. Zeile bei (0/50); jeder Buchstabe ist um einen zufälligen Wert von max. 10 Pixel nach unten versetzt.
- Die Buchstaben sollen in einem zeitlichen Abstand von 0,1 Sek. erscheinen.
- Zum Zeichnen der Buchstaben verwendet man die Methode drawChars() (siehe Java Doku: Klasse Graphics)

```
public class NervousText extends JFrame
{
    private char buchstaben[];
    private String s = "Akademie für Datenverarbeitung, Böblingen";
    private int x = 0, y = 0;

    NervousText()
    {
        super.setTitle("Nervous Text");
        buchstaben = s.toCharArray();
        setFont(new Font("TimesRoman", Font.BOLD, 36));
    }

    public void paint(Graphics g)
    {
        int zeilenabstand = 50;

        g.clearRect(0, 0, 1000, 400);

        for (int zeile = 1; zeile <= 5; zeile++)
        {
            int rotAnteil = (int) (Math.random() * 255);
            int gruenAnteil = (int) (Math.random() * 255);
            int blauAnteil = (int) (Math.random() * 255);
            Color farbe = new Color(rotAnteil, gruenAnteil, blauAnteil);
            g.setColor(farbe);
            for (int i = 0; i < buchstaben.length; i++)
            {
                x = (int) (Math.random() * 10 + 15 * i);
                y = (int) (Math.random() * 10 + 36) + zeilenabstand;
                g.drawChars(buchstaben, i, 1, x, y);
                try
                {
                    Thread.sleep(100);
                }
                catch (InterruptedException e)
                { }
            }
            zeilenabstand += 50;
        }
    }

    public static void main(String[] args)
    {
        NervousText fenster = new NervousText();
        fenster.setSize(1000, 400);
        fenster.setVisible(true);
    }
```

Animationen mit Multithreading

Ziel: Anwendung von Multithreading im Zusammenhang von Animationen, die in einer Dauerschleife laufen.

2.1. Animation „Neko“ (jap.: Kätzchen)


right1.gif


right2.gif


stop.gif


yawn.gif


scratch1.gif


scratch2.gif


sleep1.gif


sleep2.gif


awake.gif

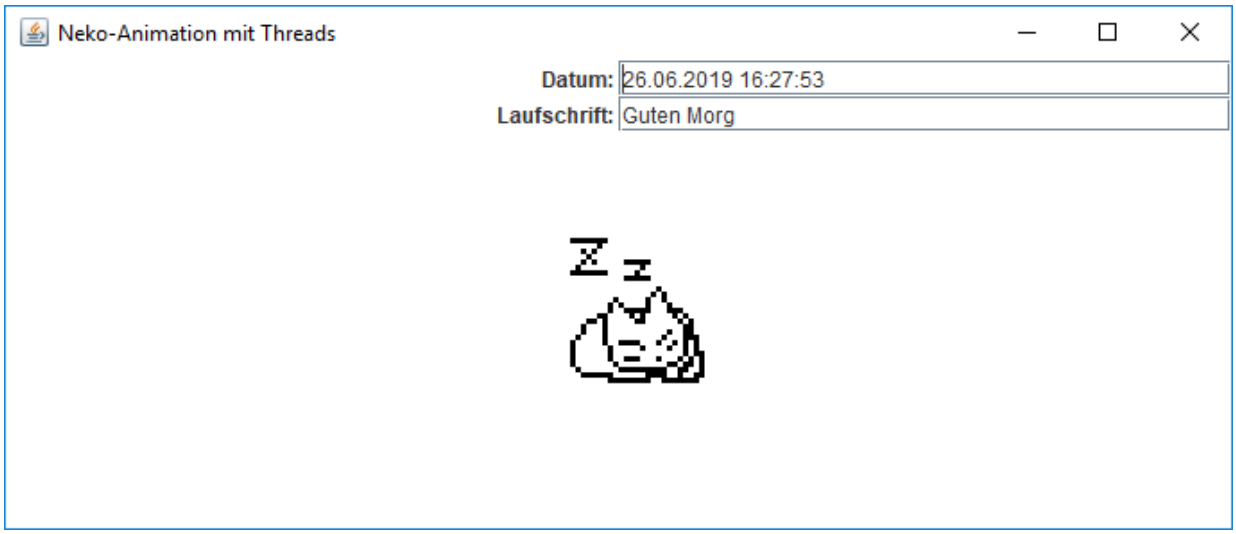
Programmieren Sie folgende GUI-Animation:

- **das Kätzchen rennt vom linken Bildschirmrand (0/50) zur Mitte (Hälfte der Fenster-breite - 50)**
  (Wechsel von 'right1.gif' und 'right2.gif' mit 0,15 Sekunden Pause – Anzeige immer um 10 Pixel nach rechts versetzt)
- **bleibt kurz stehen** ('stop.gif' – Dauer: 0,1 Sek.)
- **gähnt** ('yawn.gif' – Dauer: 0,1 Sek.)
- **kratzt sich viermal** (Wechsel von 'scratch1.gif' und 'scratch2.gif' mit 0,15 Sek. Pause)
- **schnarcht fünfmal** (Wechsel von 'sleep1.gif' und 'sleep2.gif' mit 0,25 Sek. Pause)
- **wacht auf** ('awake.gif')
- **... und rennt nach 0,1 Sek. weiter und aus dem rechten Bildschirmrand** – und taucht links wieder auf. Die Animation soll so lange laufen wie das Fenster angezeigt wird.

Hinweise:

- Überlegen Sie, wie man die Bilder – trotz ziemlich verschiedener Dateiname – in einer Schleife laden könnte. Die Bilder sollen sich ausgehend vom bin-Verzeichnis im Unterverzeichnis BilderNeko befinden.
- Da die Originalbilder ziemlich klein sind, sollen sie in dreifacher Breite und Höhe angezeigt werden: die Klasse Image besitzt die Methode `getWidth()` bzw. `getHeight()`, um die aktuelle Größe zu bestimmen (alle Bilder sind gleich groß). Man kann die Bilder dann mit der dreifachen Größe zeichnen: `drawImage()` besitzt dafür eine überlagerte Variante mit zusätzlichen Parametern (siehe Doku).
- Überlegen Sie sich, wie Sie eine mehrfache Kodierung des try/catch-Blocks bei der Anwendung von `Thread.sleep()` vermeiden können.

2.2. Das Programm von Aufgabe 2.1 soll noch erweitert werden. In einem weiteren Thread soll gleichzeitig mit der Animation noch im sekundentakt das Datum mit der Uhrzeit und als Laufschrift ein Text angezeigt werden:



Hinweise:

- Sobald der Text der Laufschrift komplett angezeigt ist, beginnt die Anzeige von neuem.
- Zur Anzeige von Datum und Uhrzeit kann man einfach jede Sekunde ein neues `LocalDateTime.Now()` ein Objekt mit der aktuellen Uhrzeit erzeugen und entsprechend formatiert (mit `DateTimeFormatter`) im `JTextField` anzeigen.

```
public class Neko_v2
{
    private JFrame frmMain;
    private NekoPanel pnlGraphik;
    private JTextField txtUhrzeit;
    private JTextField txtLaufschrift;

    // NekoPanel pnlGraphik;
    // array vom Typ Image zur Speicherung der Adressen der 17 Einzelbilder
    private Image nekoBilder[] = new Image[9];

    public Neko_v2()
    {
        this.frmMain = new JFrame("Neko-Animation mit Threads");
        this.frmMain.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.frmMain.getContentPane().setLayout(new BorderLayout());
        this.pnlGraphik = new NekoPanel(); // Objekt der inneren Klasse
        this.frmMain.getContentPane().add(pnlGraphik, BorderLayout.CENTER);

        UhrzeitPanel pnlUhrzeit = new UhrzeitPanel();
        pnlUhrzeit.setLayout(new GridLayout(0, 2));
        JLabel lblUhrzeit = new JLabel("Datum: ");
        lblUhrzeit.setHorizontalAlignment(SwingConstants.RIGHT);
        JLabel lblLaufschrift = new JLabel("Laufschrift: ");
        lblLaufschrift.setHorizontalAlignment(SwingConstants.RIGHT);
        txtUhrzeit = new JTextField(35);
        txtLaufschrift = new JTextField(35);
        pnlUhrzeit.add(lblUhrzeit);
        pnlUhrzeit.add(txtUhrzeit);
        pnlUhrzeit.add(lblLaufschrift);
        pnlUhrzeit.add(txtLaufschrift);
        this.frmMain.getContentPane().add(pnlUhrzeit, BorderLayout.NORTH);

        // Vorbereitung dafür, dass die Bilder dann in einer Schleife geladen werden
        // können
        String bilder[] = { "right1.gif", "right2.gif", "stop.gif", "yawn.gif",
                "scratch1.gif", "scratch2.gif", "sleep1.gif", "sleep2.gif", "awake.gif" };

        for (int i = 0; i < nekoBilder.length; i++)
        {
            try
            {
                nekoBilder[i] = ImageIO.read(getClass().getResource("../BilderNeko/" + bilder[i]));
            }
            catch (IOException e)
            {
                e.getMessage();
            }
        }
    }
```

```
private class NekoPanel extends JPanel  // innere Klasse von class Neko_v2
{
    private boolean bGestartet = false;

    public NekoPanel()
    {
        setBackground(Color.white);
    }

    public void paint(Graphics g) // paint-Methode wird automatisch aufgerufen, falls sie existiert
    {
        super.paint(g);

        if (!bGestartet)
        {
            MeinThread meinThread = new MeinThread(); // Objekt der Klasse MeinThread wird erzeugt
            meinThread.start(); // starten des Threads
            bGestartet = true;
        }
    }

   // innere Klasse MeinThread von class NekoPanel, abgeleitet von Standardklasse Thread
   class MeinThread extends Thread
   {
       public void run()
       {
           int xPos = 0; // x-Koordinate der linken oberen Ecke des Bildes
           int yPos = 50; // y-Koordinate der linken oberen Ecke des Bildes

           // da alle Bilder gleich groß sind, nur einmal breite und höhe für die Anzeige
           // festlegen: dreifache Breite, Höhe
           int breite = nekoBilder[0].getWidth(pnlGraphik) * 3;
           int höhe = nekoBilder[0].getHeight(pnlGraphik) * 3;
           int frmBreite;

           while (true) // andere Art von Endlosschleife, Animation soll wiederholt ablaufen
           {
               Graphics g = getGraphics();
               g.setColor(getBackground());
               frmBreite = pnlGraphik.getWidth();
               // rennen von linkem Bildrand bis Bildmitte - 50, Wechsel von right1.gif und
               // right2.gif
               for (xPos = 0; xPos < frmBreite / 2 - 50;) // keine Inkrementierung
               {
                   // Schleife für Anzeige der zwei Bilder, Pause, ...?
                   for (int i = 0; i < 2; i++)
                   {
                       g.drawImage(nekoBilder[i], xPos, yPos, breite, höhe, pnlGraphik);
                       pause(150);
                       g.fillRect(xPos, yPos, breite, höhe);
                       xPos += 10;
                   }
               }
               // stehenbleiben
               g.drawImage(nekoBilder[2], xPos, yPos, breite, höhe, pnlGraphik);
               pause(100);

               // gähnen
               g.drawImage(nekoBilder[3], xPos, yPos, breite, höhe, pnlGraphik);
               pause(100);
```

```
                    // kratzen - 4 mal
                    for (int i = 0; i < 4; i++)
                    {
                        // Schleife für Anzeige der beiden Bilder
                        for (int j = 4; j <= 5; j++)
                        {
                            g.drawImage(nekoBilder[j], xPos, yPos, breite, höhe, pnlGraphik);
                            pause(150);
                        }
                    }
                    // schnarchen - 5 mal
                    for (int i = 0; i < 5; i++)
                    {
                        for (int j = 6; j <= 7; j++)
                        {
                            g.drawImage(nekoBilder[j], xPos, yPos, breite, höhe, pnlGraphik);
                            pause(250);
                        }
                    }
                    // aufwachen()
                    g.drawImage(nekoBilder[8], xPos, yPos, breite, höhe, pnlGraphik);
                    pause(100);
                    g.fillRect(xPos, yPos, breite, höhe);

                    // weiterrennen bis zum rechten Bildrand
                    for (xPos = frmBreite / 2; xPos < frmBreite;)
                    {
                        for (int i = 0; i < 2; i++)
                        {
                            g.drawImage(nekoBilder[i], xPos, yPos, breite, höhe, pnlGraphik);
                            pause(150);
                            g.fillRect(xPos, yPos, breite, höhe);
                            xPos += 10;
                        }
                    }
                }
            }
        }

    }
    private class UhrzeitPanel extends JPanel  // innere Klasse von class Neko_v2
    {
        private boolean bGestartet = false;

        public UhrzeitPanel()
        {
            setBackground(Color.white);
        }

        public void paint(Graphics g) // paint-Methode wird automatisch aufgerufen, falls sie existiert
        {
            super.paint(g);
            if (!bGestartet)
            {
                MeinThread meinThread = new MeinThread(); // Objekt der Klasse MeinThread wird erzeugt
                meinThread.start(); // starten des Threads
                bGestartet = true;
            }
        }
```

```
        // innere Klasse MeinThread, abgeleitet von Standardklasse Thread
        class MeinThread extends Thread
        {
            public void run()
            {
                int pos = 0; // Positionszähler für array

                LocalDateTime dateTime;
                String sUhrzeit;
                String sLaufschrift = "Guten Morgen Du Schlafmütze";
                while (true) // andere Art von Endlosschleife, Animation soll wiederholt ablaufen
                {
                    dateTime = LocalDateTime.now();
                    sUhrzeit = dateTime.format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss"));
                    txtUhrzeit.setText(sUhrzeit);
                    if (pos == sLaufschrift.length() + 1)
                        pos = 1;
                    txtLaufschrift.setText(sLaufschrift.substring(0, pos));
                    pos++;
                    pause(1000);
                }
            }
        }
    }

    // Methode, um mehrmalige Codierung von try/catch zu vermeiden
    public void pause(int zeit)
    {
        try
        {
            Thread.sleep(zeit);
        }
        catch (InterruptedException e)
        {
        }
    }

    public static void main(String[] args)
    {
        Neko_v2 fenster = new Neko_v2();
        fenster.frmMain.setSize(700, 300);
        fenster.frmMain.setVisible(true);
    }
}
```