gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

Animationen ohne Multithreading

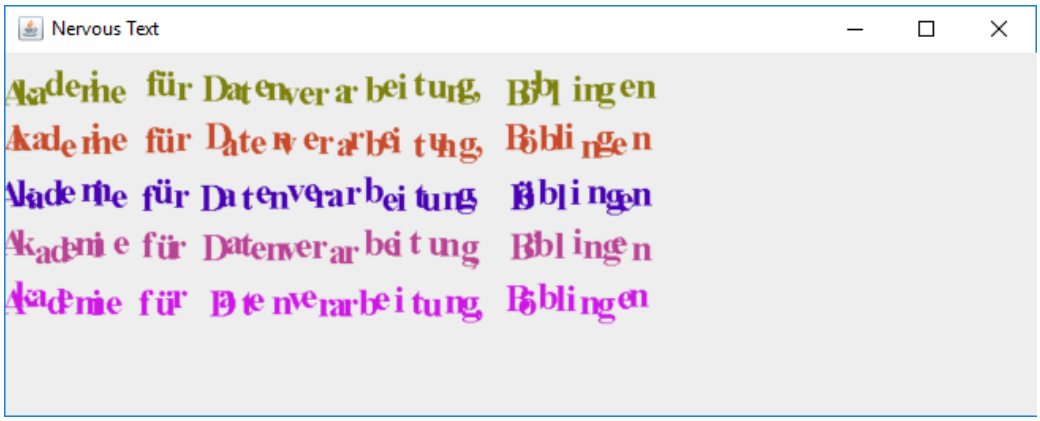
Ziel: Kennenlernen elementarer Möglichkeiten einfache Animationen in Java darzustellen und erkennen, welche Probleme sich ergeben, wenn keine Multithreading verwendet wird.

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.

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)

gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

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

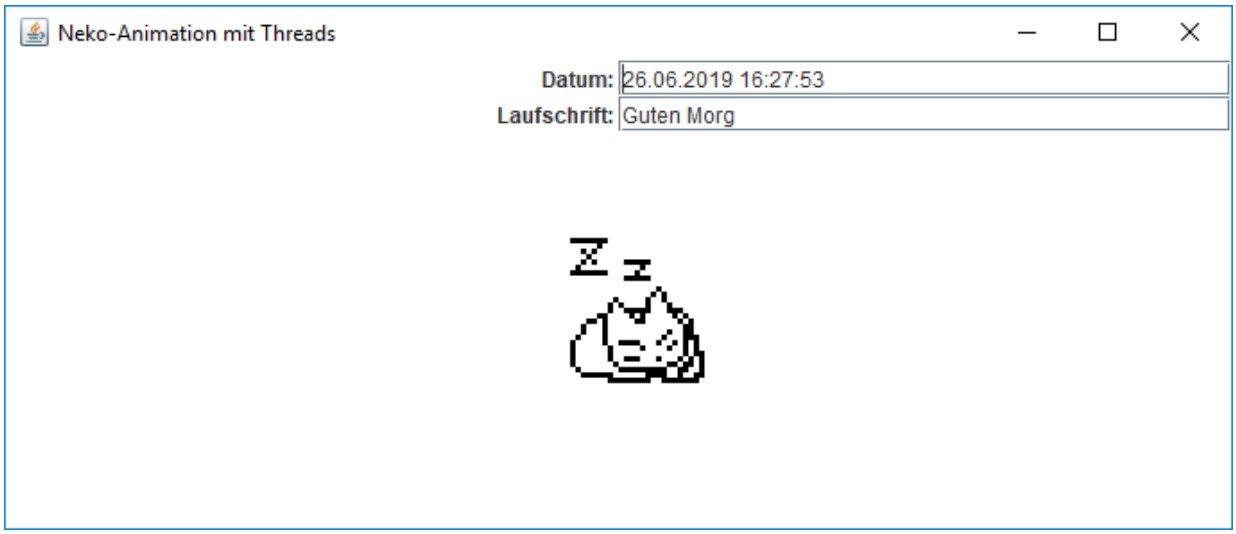
Programmieren Sie folgende GUI-Animation:

- **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.

Multithreading mit Synchronisation

Ziel: Einfache Multithreading Anwendungen mit „kritischen Abschnitten“ synchronisieren

3.1. Simulation des Ablaufs einer Spendensammelaktion[1]
Für einen guten Zweck erklären sich mehrere ehrenamtliche „Sammler“ bereit, an verschiedenen Orten Geld zu sammeln.
Es soll ein Programm erstellt werden, das den Sammelvorgang simmuliert.

- Die Namen der Spendensammler werden von der Tastatur erfasst. Es werden solange Namen eingelesen, bis der Benutzer mit <Enter> den Vorgang beendet.
- Zu jedem Spendensammler wird der Name, der gesammelte Spendenbetrag und der Start- und Endzeitpunkt der Sammelaktion gespeichert.
  Hinweis: Für eine ausreichend genaue Zeitmessung eignet sich die Methode `System.currentTimeMillis()`. Hat man zwei Zeitpunkte damit ermittelt, erhält man aus der Differenz eine Zeitdauer in Millisekunden.
- Es kann bei einer Einzelspende immer nur eine der üblichen Münzen (0,01; 0,02; 0,05; 0,1; 0,2; 0,5; 1.0; 2.0 €) gespendet werden. Bestimmen Sie den Wert der Einzelspende über Zufallswerte. Die Wahrscheinlichkeit für jede Münze soll gleichverteilt sein.
- Simulieren Sie die Zeit zwischen Einzelspenden bei einem Sammler mit zufälligen Werten im Bereich von 10 bis 100 ms (unrealistische Werte, um die Programmlaufzeit zu verkürzen). Verwenden Sie die Thread.Sleep() Methode[2].
- Sobald ein Spendensammler mehr als 100 € gesammelt hat, beendet er seine Sammelaktion.
- Wenn alle Sammler fertig sind, soll eine Bildschirmausgabe nach folgendem Muster erscheinen:

```
<terminated> Spenden_UI [Java Application] C:\Program Files\Java\jdk1.8.0_191\bin\javaw.exe (28.06.2019, 16:31:55)
Sammlername (Abbruch mit <Enter>): Hans
Sammlername (Abbruch mit <Enter>): Helmut
Sammlername (Abbruch mit <Enter>): Herbert
Sammlername (Abbruch mit <Enter>): Hugo
Sammlername (Abbruch mit <Enter>):

Sammelaktion beendet!
Name: Hans       Dauer:    23,3970 s  Betrag:  100,01 €
Name: Helmut     Dauer:    19,6590 s  Betrag:  100,22 €
Name: Herbert    Dauer:    21,6780 s  Betrag:  100,92 €
Name: Hugo       Dauer:    21,7210 s  Betrag:  100,40 €
```

[1] Aufgabenidee aus: Volker Janßen „Angewandte Informatik Java/Softwareentwicklung“ Feb. 2018 Eigenverlag
[2] Hinweis: Simulationen für die exakte Untersuchung von Abläufen, wie z.B. die Auslastung von Verkehrs- oder Kommunikationssystemen führt man im Allgemeinen unabhängig von der Realzeit aus, da sonst die Prozessorauslastung zu Verfälschungen führen könnte. Lediglich für eine gleichzeitige Visualisierung der Abläufe wäre eine Lösung wie in unserer Programmierübung sinnvoll.

3.2. Version 2 „Spendensammelaktion“

Die Simulation von Aufgabe 3.1 soll so verändert werden, dass alle Spendensammler ihre Sammelaktion einstellen, sobald ein Gesamtspendenbetrag von über 500 € erreicht wurde. Ausgabe:

```
Name: Hans       Dauer:    30,6190 s Betrag:  119,77 €
Name: Hugo       Dauer:    30,5940 s Betrag:  126,67 €
Name: Helmut     Dauer:    30,6090 s Betrag:  134,23 €
Name: Herbert    Dauer:    30,5770 s Betrag:  120,01 €
                               Gesamtbetrag:  500,68 €
```

- Überlegen Sie, welche Größe in dieser Aufgabe die gemeinsam zu benutzende Größe ist.
- Legen Sie für diese Größe eine Klasse an, ähnlich zu der Klasse „Fass“ von dem Bsp. im Skript (30_Threads Abs. 2.4).
- Setzen Sie für die schreibend zugreifende Methode das `sychronized` Schlüsselwort.
- Bis das Programm läuft, ist es geschickter für den Betrag statt 500 z.B. 50 einzusetzen, um beim Testen nicht so lange warten zu müssen. Allerdings sollte man dann das Programm am Ende mit dem Betrag 500 testen, da man erst bei dieser Größe einen Effekt der Synchronisation erkennen kann. Am besten man testet auch einmal ohne Synchronisation und prüft, ob die Summe der Einzelbeträge mit der Gesamtsumme übereinstimmt.

3.3. Version 3 „Spendensammelaktion“

Die Simulation von Aufgabe 3.1 soll so verändert werden, dass alle Spendensammler ihre Sammelaktion einstellen, sobald eine Gesamt-Sammelzeit von über z.B. 20 s erreicht wurde. Die Sammler-Threads sollen auch dann gestoppt werden, wenn sie sich gerade im Sleep Zustand befinden.
Hinweis: Die Wahrscheinlichkeit, dass ein Sammler-Thread sich im Sleep-Zustand befindet ist sehr groß. Erkundigen Sie sich in der Oracle Doku, wie sich die Thread.Sleep Methode verhält, wenn ein Thread eine Interrupt-Nachricht erhält. Gegebenenfalls soll der Sleep Zustand unterbrochen werden, und der Thread sich beenden. Ausgabe:

```
Sammelaktion beendet!
Name: Hans       Dauer:     4,0680 Betrag:   22,10
Name: Hugo       Dauer:     4,0620 Betrag:   20,19
Name: Helmut     Dauer:     4,0620 Betrag:   24,30
Name: Herbert    Dauer:     4,0620 Betrag:   20,76
Name: Harmut     Dauer:     4,0620 Betrag:   19,42
Gesamtbetrag: 106,77 € Gesamtdauer:    20,2990 s
```