Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 2 SoSe 2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof R. Lorenz, L. Petrak, J. Linne, V. Le Claire, P. Schalk 19.07.2024

Lösungsvorschlag zu Übungsblatt 12

- Dieses Übungsblatt ist **unbewertet** und wird nicht korrigiert.
- Es ist also nicht abzugeben.
- Alle Aufgaben dieses Übungsblatts sind **klausurrelevant**.

Aufgabe 45 ** *(Threads implementieren, 22 Minuten)*

a) *(**, alte Klausuraufgabe, 12 Minuten)*
Implementieren Sie ein Fenster `WaitFrame` der Größe `100 x 100`, das immer im Wechsel für jeweils 1 Sekunde die Zeichenfolge `wait` anzeigt und dann nicht anzeigt. Drückt der Benutzer die Taste `q`, soll dieser Verarbeitungsprozess gestoppt und das Fenster geschlossen werden.

Das Fenster soll keine weiteren Komponenten enthalten und auf keine weiteren Ereignisse reagieren.
Lösung:

```
public class WaitFrame extends JFrame {
    public WaitFrame() {
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        JLabel l = new JLabel("wait");
        add(l);
        Thread t = new Thread(() -> {
            while (!Thread.interrupted()) {
                try {
                    Thread.sleep(1000);
                    if (l.isVisible())
                        l.setVisible(false);
                    else
                        l.setVisible(true);
                } catch (InterruptedException e) {
                    break;
                }
            }
        });
        addKeyListener(new KeyAdapter() {
            @Override
            public void keyTyped(KeyEvent e) {
                if (e.getKeyChar() == 'q') {
                    t.interrupt();
                    dispose();
                }
            }
        });
        t.start();
        setSize(100, 100);
        setVisible(true);
    }
}
```

b) *(**, alte Klausuraufgabe, 10 Minuten)*
Implementieren Sie einen Verarbeitungsprozess `Counter`, der in Abständen von 1 Sekunde die positiven ganzen Zahlen (beginnend mit der Zahl `1`) aufsteigend auf Kommandozeile ausgibt. Der Verarbeitungsprozess soll von außen gestoppt und wieder bei `1` gestartet werden können.

Achten Sie darauf, dass in der `Counter`-Klasse maximal ein Thread gleichzeitig läuft.
Lösung:

```
public class Counter implements Runnable {
    private int count;
    private Thread countThread;

    public Counter() {
        start();
    }

    public synchronized void stop() {
        countThread.interrupt();
        countThread = null;
    }

    public synchronized void start() {
        if (countThread == null) {
            count = 1;
            countThread = new Thread(this);
            countThread.start();
        }
    }

    @Override
    public void run() {
        while (true) {
            try {
                System.out.println(count++);
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                break;
            }
        }
    }
}
```

Aufgabe 46 ** *(Verzahnung und Synchronisation von Threads, 15 Minuten)*

a) *(**, alte Klausuraufgabe, 7 Punkte)*
Betrachten Sie den folgenden Programmcode und beantworten Sie die nachfolgenden Fragen:

```
public class IntValue {
    public int value = 0;

    public void inc() {
        value += 1;
    }

    public void dec() {
        value -= 10;
    }

    public void print() {
        System.out.print(value);
    }
}

public class Thread1 extends Thread {
    private IntValue iv;

    public Thread1(IntValue iv) {
        this.iv = iv;
    }

    public void run() {
        iv.inc();
    }
}

public class Thread2 extends Thread {
    private IntValue iv;

    public Thread2(IntValue iv) {
        this.iv = iv;
    }

    public void run() {
        iv.dec();
    }

}

public class IntValueMain {
    public static void main(String[] args) {
        IntValue v = new IntValue();
        Thread1 t1 = new Thread1(v);
        Thread2 t2 = new Thread2(v);
        t1.start();
        t2.start();
        v.print();
    }
}
```

(i) Welche sind die möglichen Kommandozeilenausgaben? (Ohne Begründung)
Lösung:
Die möglichen Ausgaben sind:

- 0
- 1
- -10
- -9

(ii) Wie kann man den Programmcode ohne Verwendung von `volatile` verbessern, so dass nur noch die Ausgabe `-9` möglich ist?
Lösung:

- Der `main`-Thread muss mit der Ausgabe warten, bis die Threads `t1` und `t2` beendet sind: Aufruf von `t1.join()` und `t2.join()` nach Start der beiden Threads.
- Die beiden Threads `t1` und `t2` dürfen sich beim Schreiben nicht verzahnen: Definiere die beiden Methoden `inc()` und `dec()` als `synchronized`.

b) *(**, 8 Minuten)*
Betrachten Sie eine `int`-Variable `i`, die zu Beginn den Wert `0` hat, und folgende Sequenz von Anweisungen:

a) `i = i + 1;`

b) `System.out.print(i);`

Nehmen Sie an, zwei Threads werden asynchron gestartet, führen jeweils diese beiden Anweisungen bzgl. derselben Variable `i` aus und arbeiten dabei jeweils auf einer lokalen Kopie von `i`. Zu welchen Ausgaben kann es dann kommen? Begründen Sie Ihre Antwort.

Lösung:
Die möglichen Ausgaben sind:

- `11`: Beide Threads lesen den alten Wert `0` von `i`, bevor der jeweils andere Thread den Wert erhöht. Beide erhöhen dann `0` um `1` und geben den Ergebniswert aus.
- `12`: Der erste Thread erhöht den Wert von `i` auf `1` und gibt diesen aus, bevor der zweite Thread den Wert von `i` liest, um `1` auf `2` erhöht und ausgibt.
- `21`: Der Wert der lokalen Kopie des Threads mit Wert `2` für `i` wird nach dem Wert der lokalen Kopie des anderen Threads mit Wert `1` für `i` ausgegeben.

Aufgabe 47 ** *(Threads implementieren, 20 Minunten)*

Betrachten Sie die im Digicampus zusätzlich zur Angabe gegebene Hauptanwendungsfenster `MemoryTrainingFrame`. Das Fenster soll in jeder Runde eine längere Folge von Ziffern generieren und diese über die Hintergrundfarbe der 9 Buttons anzeigen. Danach soll der Nutzer die Buttons in derselben Reihenfolge anklicken – gelingt ihm das, so bekommt er einen Punkt, ansonsten wird der Punktestand auf 0 zurückgesetzt.

a) *(**, 10 Minuten)*
Implementieren Sie die Methode `run`. Darin soll

- eine zufällige, ganze Zahl zwischen 0 (einschließlich) und 9 (ausschließlich) generiert und zur Liste `solution` hinzugefügt werden,
- der Reihe nach für jede Zahl in `solution` die Hintergrundfarbe des Button im Array `buttons` mit dem entsprechenden Index blau und nach einer Sekunde wieder normal gefärbt werden[1],
- bei einer Unterbrechung der Tread `thread` unterbrochen werden,
- am Ende
  - das Klicken aller Buttons mit Ziffern erlaubt und
  - `thread` auf `null` gesetzt werden.

Lösung:

```
public void run() {
    Random rng = new Random();
    solution.add(rng.nextInt(9));
    try {
        for (int i = 0; i < solution.size(); i++) {
            buttons[solution.get(i)].setBackground(Color.BLUE);
            Thread.sleep(1000);
            buttons[solution.get(i)].setBackground(null);
        }
    } catch (InterruptedException e) {
        thread.interrupt();
    }
    setButtonsEnabled(true);
    thread = null;
}
```

b) *(**, 5 Minuten)*
Implementieren Sie die Methode `startRound`. Beim Starten einer Runde sollen

- Alle Ziffern-Buttons deaktiviert werden,
- der Start-Button deaktiviert werden und
- ein neuer Thread, der das aktuelle Objekt ausführt, gestartet werden.

Stellen Sie sicher, dass immer nur ein Thread gleichzeitig laufen kann.
Lösung:

```
private void startRound() {
    if (thread == null) {
        startButton.setEnabled(false);
        setButtonsEnabled(false);
        thread = new Thread(this);
        thread.start();
    }
}
```

[1] Mit dem Parameter `null` in der Methode `setBackground` kann die Hintergrund-Farbe des Containers gewählt werden

c) *(**, 5 Minuten)*
Implementieren Sie die Methode `onExit`. Wenn ein Thread vorhanden ist, soll dieser unterbrochen werden. Dann soll das Fenster geschlossen und die Anwendung beendet werden.
Lösung:

```
private void onExit() {
    if (thread != null) {
        thread.interrupt();
    }
    dispose();
    System.exit(0);
}
```

Aufgabe 48 ** *(Threads implementieren, 25 Minunten)*

a) *(**, 11 Minuten (alte Klausuraufgabe))*
Erweitern Sie das Hauptprogramm der gegebenen Klasse `SumCalculator` und die Klasse selbst so, dass eine asynchrone Berechnung von Summen ermöglicht wird.

- Sorgen Sie dafür, dass Objekte der Klasse `SumCalculator` von einem Thread ausgeführt werden können.
- Wenn ein `SumCalculator`-Objekt ausgeführt wird, soll die Summe des `int`-Array-Attributs berechnet und in dem Attribut `sum` gespeichert werden.
- Das Programm soll zwei `SumCalculator`-Objekte zur Berechnung der Summe der beiden Arrays `set1` und `set2` erstellen. Jedes Objekt soll von einem eigenen Thread ausgeführt werden.
- Das Programm soll warten, bis beide Threads ihre Berechnungen abgeschlossen haben.
- Danach soll die Summe der beiden Endergebnisse berechnet und auf der Kommandozeile ausgegeben werden.
- Beim Auftreten von Ausnahmen soll ihr Stack Trace ausgegeben werden.

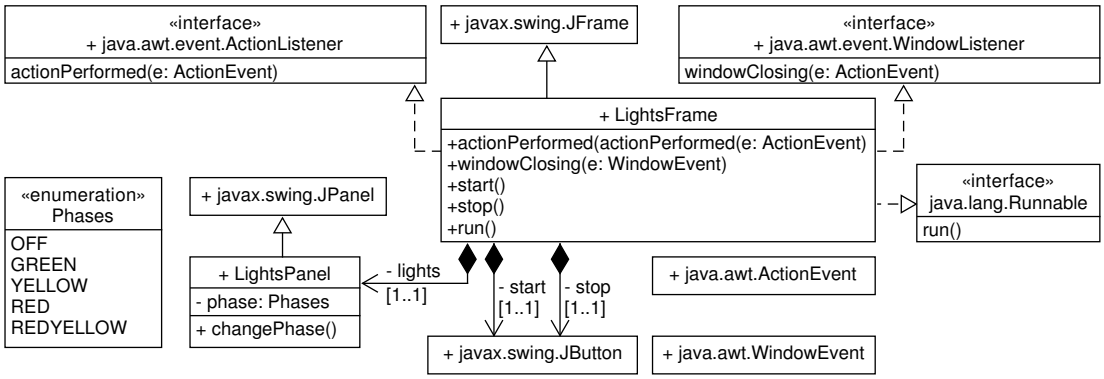
Lösung:

```
public class SumCalculator implements Runnable {
    public static final int[] set1 = {1, 2, 3, 4, 5};
    public static final int[] set2 = {6, 7, 8, 9, 10};

    private int[] numbers;
    private int sum;

    public SumCalculator(int[] numbers) {
        this.numbers = numbers;
        this.sum = 0;
    }

    public int getSum() {
        return sum;
    }

    @Override
    public void run() {
        for (int i = 0; i < numbers.length; i++) {
            sum += numbers[i];
        }
    }

    public static void main(String[] args) {
        SumCalculator calculator1 = new SumCalculator(set1);
        SumCalculator calculator2 = new SumCalculator(set2);

        Thread thread1 = new Thread(calculator1);
        Thread thread2 = new Thread(calculator2);
        thread1.start();
        thread2.start();
```

```
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        int sum1 = calculator1.getSum();
        int sum2 = calculator2.getSum();
        int totalSum = sum1 + sum2;
        System.out.println("Total Sum: " + totalSum);
    }
}
```

- Eine Lösung mit 'extends Thread' wäre auch möglich

b) *(**, 14 Minuten (alte Klausuraufgabe))*
Gegeben sei ein Java-Programm, dessen Struktur Sie dem folgenden UML-Klassendiagramm entnehmen können:



Die Klassen für die Darstellung aller grafischen Elemente und die Methode `changePhase` zum Wechsel der Ampelphase liegen bereits fertig implementiert vor – es fehlt nur die Ereignisbehandlung. Implementieren Sie daher die Klasse `LightFrame` , und setzen Sie die folgende Funktionalität zur Schaltung der verschiedenen Phasen einer Ampel um:

- Über einen Klick auf die Schaltfläche `start` soll das automatisierte Wechseln der Ampelphase mit einer Wartezeit von jeweils einer Sekunde gestartet werden können.
- Über die Schaltfläche `stop` soll das automatische Wechseln gestoppt werden können.
- Wenn das automatisierte Wechseln bereits aktiv ist, soll kein weiterer Vorgang gestartet werden können.
- Beim Schließen des Fensters soll ein evtl. noch aktiver Wechselvorgang gestoppt werden.

Lösung:

```
public class LightsFrame extends JFrame
        implements ActionListener, WindowListener, Runnable {
    private Thread thread;

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == start)
            start();
        else
            stop();
    }
```

```
        private void start() {
                if (thread == null) {
                        thread = new Thread(this);
                        thread.start();
                }
        }

        private void stop() {
                if (thread != null) {
                        thread.interrupt();
                        thread = null;
                }
        }

        @Override
        public void run() {
                while (!thread.isInterrupted()) {
                        try {
                                Thread.sleep(1000);
                                lights.changePhase();
                        } catch (InterruptedException e) {
                                break;
                        }
                }
        }

        @Override
        public void windowClosing(WindowEvent e) {
                stop();
        }
}
```

- Alternativ auch Implementierung als gesonderte `Thread`-Klasse möglich
- Auch alternative Lösungen zum Erstellen / neustarten von Threads möglich