

---

## 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.

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.

---

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

---

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.

### Aufgabe 47 \*\* (*Threads implementieren, 20 Minuten*)

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<sup>1</sup>,
- bei einer Unterbrechung der Tread `thread` unterbrochen werden,
- am Ende
  - das Klicken aller Buttons mit Ziffern erlaubt und
  - `thread` auf `null` gesetzt werden.

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.

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.

---

<sup>1</sup>Mit dem Parameter `null` in der Methode `setBackground` kann die Hintergrund-Farbe des Containers gewählt werden

---

## Aufgabe 48 \*\* (*Threads implementieren, 25 Minuten*)

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.

```
public class SumCalculator {
    private int[] numbers;
    private int sum;

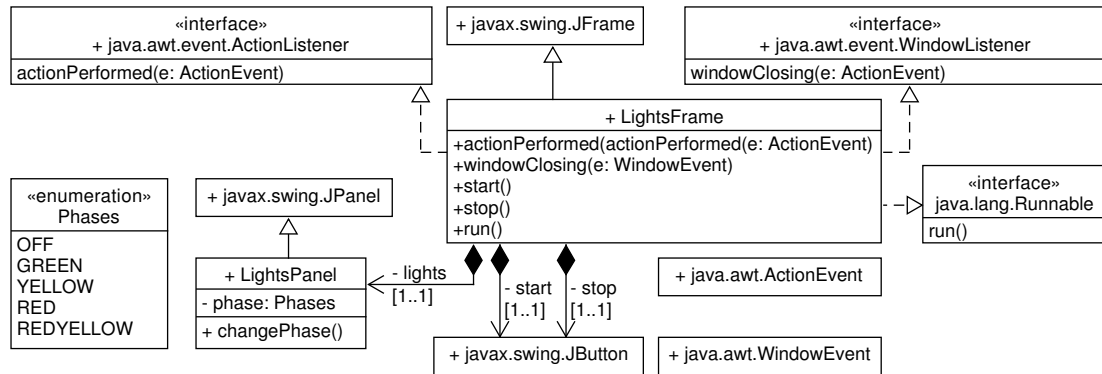
    public SumCalculator(int[] numbers) {
        this.numbers = numbers;
        this.sum = 0;
    }

    public int getSum() {
        return sum;
    }

    public static void main(String[] args) {
        int[] set1 = {1, 2, 3, 4, 5};
        int[] set2 = {6, 7, 8, 9, 10};
    }
}
```

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.