HSLU Hochschule Luzern

Algorithmen & Datenstrukturen (AD)

Übung: Threads & Synchronisation (N1)

Themen: Erzeugen und Starten von Threads, Beenden von Threads, Elementare Synchronisationsmechanismen, Synchronisation durch Monitor-Konzept
Zeitbedarf: ca. 240min.

Roger Diehl, Version 1.0 (FS 2025)

1 Ballspiele (max. 90' inkl. optionaler Teil)

1.1 Lernziele

- Threads erzeugen und beenden
- Zugriff auf gemeinsame Ressourcen

1.2 Grundlagen

Diese Aufgabe basiert auf dem AD Input N11. Es ist eine «Fingerübung» zu einfachen Threads. Wenden Sie nicht mehr Zeit auf als oben angegeben! Zudem erhalten Sie die Code Vorlage `ch.hslu.ad.exercise.n1.balls`.

1.3 Aufgabe

Ihre Aufgabe ist es mit Hilfe von Java ein paar bunte Bälle zu produzieren und diese der Schwerkraft auszusetzen. Benutzen Sie dazu die gegeben Klassen `Canvas` und `Circle`. Die beiden Klassen stammen aus dem OOP BlueJ Code. Das Programm soll eine 2D Grafik erstellen und muss folgende Eigenschaften aufweisen:

- Die Bildgrösse beträgt 600x400 Punkte (Standardeinstellung). Sie können aber auch eine andere Grösse in der Klasse `Canvas` einstellen.
- Der Ballradius beträgt zufällig zwischen 20 und 50 Punkten.
- Die Farbe des Balls ist zufällig aus der gegebenen Palette: `"red", "black", "blue", "yellow", "green", "magenta"`.
- Nach dem Erzeugen fällt der Ball nach unten, am unteren Bildrand „platzt“ er, d.h. der Ball verschwindet.
- Die Fallgeschwindigkeit des Balls ist zufällig.
- Erzeugen Sie viele Bälle beim Start des Programms. Probieren Sie beide Thread-Varianten aus, konventionelle und virtuelle Threads.
- **Optional**: Ein Ball wird mit Drücken der (linken und/oder rechten) Maustaste generiert. Die Position des Balls ist beim Start an der x-y-Position der Maus.

Wichtig: Die Ausgabe und Bewegung der Bälle können, bzw. dürfen flackern. Ziel der Aufgabe ist nicht die perfekte grafische Applikation, sondern das «Leben» eines Balles zu implementieren.

Tipps:

- Analysieren Sie zuerst die Klassen `Canvas` und `Circle`.
- Erstellen Sie für den Ball einen Task.
- Das „Leben“ eines Balles wird durch einen Thread „gelebt“.
- Falls Exceptions auftreten, dürfen Sie die Klassen `Canvas` und `Circle` gerne ändern.

Wie die „Ballspiele“ aussehen könnten, zeigt Ihnen dieses Bild.

1.4 Reflektion

Reflektieren Sie die Aufgabe (hilft auch bei einer eventuellen Präsentation) und beantworten Sie sich die folgenden Fragen:

- Warum eigentlich Threads verwenden?
- Wie werden die Threads erzeugt?
- Wann werden die Threads beendet?
- Merken Sie einen Unterschied zwischen konventionellen und virtuellen Threads?
- Was ist die gemeinsame Ressource in dieser Aufgabe?

2 Bankgeschäfte (ca. 90')

2.1 Lernziele

- Threads erzeugen
- Auf das Ende von Threads warten
- Zugriff auf gemeinsame Ressourcen

2.2 Grundlagen

Diese Aufgabe basiert auf dem AD Input N11 und N12. Sie erhalten die Code Vorlage `ch.hslu.ad.exercise.n1.bank`. Zudem können Sie Ihre Implementation aus der Übung D1 nutzen. Lesen Sie zuerst die ganze Aufgabe durch.

2.3 Aufgabe

Bei dieser Aufgabe ist es wichtig, dass Sie die einzelnen Punkte a), b), c) und d) nacheinander abarbeiten, auch wenn Sie die Lösung schon zu Beginn sehen. Nur so können Sie den Aha-Effekt erleben, den diese Aufgabe erzielen will.

Anmerkung: Es hat sich gezeigt, dass ChatGPT diese Aufgabe nicht richtig versteht und deshalb bei der Lösungsfindung falsch vorgeht. Nutzen Sie die Zeit, indem Sie diese Aufgabe selbst lösen.

Gegeben ist die folgende Bankkonto Klasse.

```java
/**
 * Einfaches Bankkonto, das nur den Kontostand beinhaltet.
 */
public final class BankAccount {

    private int balance;

    /**
     * Erzeugt ein Bankkonto mit einem Anfangssaldo.
     * @param balance Anfangssaldo
     */
    public BankAccount(final int balance) {
        this.balance = balance;
    }

    /**
     * Erzeugt ein Bankkonto mit Kontostand Null.
     */
    public BankAccount() {
        this(0);
    }
```

```java
    /**
     * Gibt den aktuellen Kontostand zurück.
     * @return Kontostand.
     */
    public int getBalance() {
        return this.balance;
    }

    /**
     * Addiert zum bestehen Kontostand einen Betrag hinzu.
     * @param amount Einzuzahlender Betrag
     */
    public void deposite(final int amount) {
        this.balance += amount;
    }

    /**
     * Überweist einen Betrag vom aktuellen Bankkonto an ein Ziel-Bankkonto.
     * @param target Bankkonto auf welches der Betrag überwiesen wird.
     * @param amount zu überweisender Betrag.
     */
    public void transfer(final BankAccount target, final int amount) {
        this.balance -= amount;
        target.deposite(amount);
    }
}
```

a) Mit Instanzen dieser Klasse sollen Sie Überweisungen tätigen. Sie können sich vorstellen, dass im Bankalltag weltweit tausende von Überweisung pro Sekunde stattfinden. Auch sind Zugriffe denkbar, die gleichzeitig auf ein Konto gemacht werden. Diesen Sachverhalt wollen wir simulieren. Gegeben ist folgendes Szenario (`ch.hslu.ad.exercise.n1.bank`):

- Es gibt eine Liste von Quell Konten und eine Liste von Ziel Konten. Optional: Nutzen Sie Ihre eigene Listen Implementation aus der Übung D.
- Ein (grosser) Betrag, am besten immer in gleicher Höhe, soll von den Quell Konten an die Ziel Konten überwiesen werden und wieder zurück.
- Die Überweisung von der Quelle zum Ziel wird von einem Bankauftrag (Klasse `AccountTask`) mit Hilfe eines Threads gemacht.
- Die Rück-Überweisung vom Ziel zur Quelle wird von einem neuen Bankauftrag (Klasse `AccountTask`) mit Hilfe eines anderen Threads gemacht.
- Damit ein sehr grosser Betrag bei der Überweisung nicht auf dem Radar der Finanzaufsichtsbehörde erscheint, wird die Überweisung in Micro-Überweisungen mit jeweils sehr kleinen Beträgen aufgeteilt.
- Starten Sie die Bankaufträge mit der `DemoBankAccount` Klasse:

```java
final Thread[] threads = new Thread[number * 2];
for (int i = 0; i < number; i++) {
    threads[i] = new Thread(new AccountTask(
        sourceBankAccounts.get(i), targetBankAccounts.get(i), amount));
    threads[i + number] = new Thread(new AccountTask(
        targetBankAccounts.get(i), sourceBankAccounts.get(i), amount));
}
for (final Thread thread : threads) {
    thread.start();
}
```

- Die Ausgabe der Liste der Quell und Ziel Konten sollte wie folgt aussehen, nachdem alle Bankaufträge durchgeführt, bzw. die Threads beendet wurden.

```
...
2025-03-13 10:17:48,496 INFO  - Bank accounts after transfers
2025-03-13 10:17:48,496 INFO  - source(0) = 100000; target(0) = 0;
2025-03-13 10:17:48,496 INFO  - source(1) = 100000; target(1) = 0;
2025-03-13 10:17:48,506 INFO  - source(2) = 100000; target(2) = 0;
2025-03-13 10:17:48,506 INFO  - source(3) = 100000; target(3) = 0;
2025-03-13 10:17:48,506 INFO  - source(4) = 100000; target(4) = 0;
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 1.910 s
Finished at: 2025-03-13T10:17:48+01:00
Final Memory: 7M/30M
------------------------------------------------------------------------
```

Experimentieren Sie mit verschiedenen Einstellungen:

- Anzahl der Bankkonten (Grösse der Liste)
- Anzahl Bankaufträge (Threads)
- Höhe des zu überweisenden Betrages
- Anzahl Micro-Überweisungen

Reflektion:

- Was sollte beim Szenario passieren, wenn das Programm korrekt ablaufen würde?
- Was beobachten Sie?
- Wie erklären Sie sich Programmverhalten?

b) Analysieren Sie die Bankkonto Klasse und identifizieren Sie die Schwachstelle. Wie können Sie ein noch stärkeres Fehlverhalten provozieren?

c) Eliminieren Sie die Schwachstelle mit Hilfe des elementaren Synchronisations-mechanismus wie in Folie 9 aus N12_IP_Synchronisation gezeigt. Welche Art von Synchronisation setzen Sie ein (Instanz oder Klasse)?

Reflektion:

- Welche Art von Synchronisation ist für die Bankkonto Klasse besser? Warum?
- Was beobachten Sie nun?
- Wie erklären Sie sich Programmverhalten?

d) Eliminieren Sie die Schwachstelle nun vollständig, falls immer noch Fehler passieren. Wie machen Sie das am besten? (Tipp: Folie 20 aus N12_IP_Synchronisation)

2.4 Reflektion

Reflektieren Sie die Aufgabe (hilft auch bei einer eventuellen Präsentation) und beantworten Sie sich die folgenden grundsätzlichen Fragen (die Antworten gelten nicht nur für Java):

- Wie müssen Zugriffe auf gemeinsame Ressourcen am besten geschützt werden?
- Was sollte bei der Synchronisation in jedem Fall vermieden werden?
- Was verursacht der Einsatz von Synchronisation im Allgemeinen?

3 JoinAndSleep (ca. 60')

3.1 Lernziele

- Threads erzeugen
- Auf Threads warten
- Threads unterbrechen

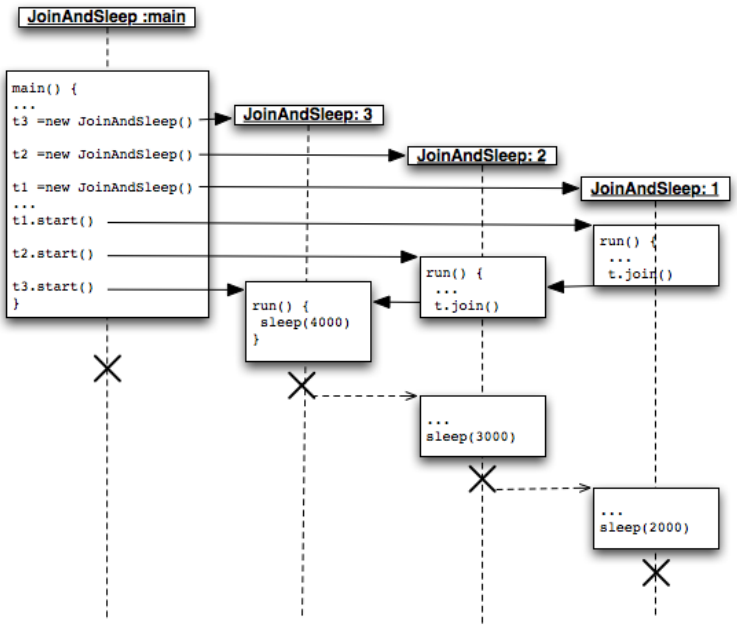
3.2 Grundlagen

Diese Aufgabe basiert auf dem AD Input N11. Zudem erhalten Sie die Code Vorlage `ch.hslu.ad.exercise.n1.joins`.

3.3 Aufgabe

Ziel der Aufgabe ist es drei Threads zu programmieren die auf das Beenden des jeweils anderen Threads warten und dann eine Zeit schlafen:

1) Jeder neue Zustand wird auf die Konsole ausgegeben (`LOG.info`).
2) Als erstes nach dem Start wartet der Thread bis der Ziel-Thread, auf den er referenziert, beendet ist. Ist kein Ziel-Thread referenziert, so geht der Thread sofort über zum nächstens Schritt.
3) Die Threads schlafen für eine vorgegebene Zeit in Millisekunden.
4) Die Threads beenden ordentlich.

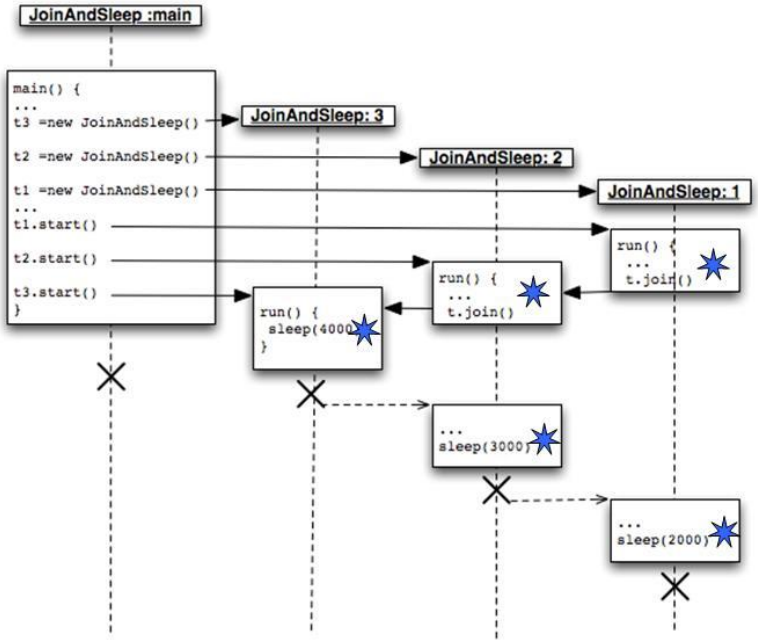


a) Die Demo Applikation soll folgende Aktionen ausführen, wie im obigen Bild gezeigt:

- Erzeugen von Thread 3: Er soll auf keinen Thread warten und dann 4000ms schlafen
- Erzeugen von Thread 2: Er soll auf Thread 3 warten und dann 3000ms schlafen
- Erzeugen von Thread 1: Er soll auf Thread 2 warten und dann 2000ms schlafen
- Start von Thread 1
- Start von Thread 2
- Start von Thread 3

Tipp: Benutzen Sie die `JoinAndSleepTask` Klasse, um die Aktionen konfigurieren zu können.

b) Erweitern Sie die Demo Applikation, so dass ein Thread zu einem beliebigen Zeitpunkt unterbrochen werden kann. Der Thread, bzw. Task, muss den Abbruch dokumentieren, in welchen Abschnitt – Sleep oder Join – er unterbrochen wurde.



✸ mögliche Unterbrechungspunkte

3.4 Reflektion

Reflektieren Sie die Aufgabe und beantworten Sie sich die folgenden grundsätzlichen Fragen:

- Worin lag die grösste Herausforderung in dieser Aufgabe?
- Wenn Sie die Aufgabe (und mögliche Lösungen) «gegoogelt» haben. Wo liegt der grösste Unterschied zu den Aussagen im AD Input N11?
- In welchem Zustand ist ein Thread, der auf einen anderen Thread wartet?
- Welcher Programmteil, bzw. Thread, muss die laufenden Threads abbrechen?