Übungen 5

Patrick Bucher

26.03.2017

Inhaltsverzeichnis

1 Ballspiele

Warum überhaupt Threads verwenden?

Würden die Bälle synchron animiert, könnte das User-Interface beim Zeichnen keine Click-Events mehr entgegennehmen. Man könnte also, bei vielen zu zeichnenden Bällen, nicht mehr zuverlässig weitere Bälle hinzufügen.

Warum es für jeden Ball einen Thread geben soll, erschliesst sich mir jedoch nicht. Bei meinem Beispiel führt dies dazu, dass jeder Ball periodisch die ganze Zeichenfläche neu aufbauen lässt. Das ist ineffizient und hat den Nebeneffekt, dass die Animation bei vielen Bällen flüssiger läuft als bei wenigen, weil dann mehr Aktualisierungen nötig sind.

Würde man nur einen Zeichen-Thread verwenden, müsste weniger aktualisiert werden.

Wie werden die Threads erzeugt?

Die Methode `createBall()` wird von einem Click-Event ausgelöst:

```
public void createBall(int x, int y) {
    int radius = getRandomNumber(20, 50);
    Color color = getRandomColor();
    Ball ball = new Ball(x, y, radius, color, this);
    balls.add(ball);
    ball.start();
    System.out.println("added ball, now " + balls.size() + " balls");
}
```

Wann “sterben” die Treads?

Das “Leben” eines Balls hat zwei Phasen: das Fallen und das Erblassen. Sobald diese beiden Phasen durchschritten sind, läuft die Schleife in der `run()`-Methode des Balls nicht mehr weiter.

Wird ein Thread unterbrochen, wird das `interrupted`-Flag auf `true` gesetzt, wodurch auch die Schleife in der `run()`-Methode unterbrochen wird.

Die `paintComponent()`-Methode der Zeichenfläche prüft, ob der jeweils zu zeichnende Ball-Thread schon "erledigt", d.h. unterbrochen oder abgearbeitet, ist. Ist das der Fall, wird er aus der Collection mit zu zeichnenden Bällen entfernt.

Wie wird die Darstellung der Bälle aktualisiert?

Jeder Ball ist zugleich ein Thread, der in einer Schleife die Zeichenfläche aktualisieren lässt und danach einige Millisekunden wartet (`Thread.sleep(10)`). Dazu ruft er die Methode `repaint()` von `JPanel` (bzw. dessen Vererbung `DrawingArea`) auf.

2 Hello World

Benötigen Sie überhaupt einen oder mehrere Threads?

Ich benötige einen Thread. Ohne zusätzlichen Thread wäre die Benutzeroberfläche ständig mit Zeichnen und Warten beschäftigt und könnte so keine Klicks entgegennehmen.

Welche Thread-Implementierung verwenden Sie?

Ich erbe von der Klasse `Thread` und implementiere das Interface `Runnable`. Das ist die einfachste Lösung für eine einfache Aufgabe. Eine Alternative dazu wäre der `SwingWorker`.

Wo starten Sie den/die Thread(s)?

Ich starte den Thread, wenn das Fenster komplett geladen ist. Dazu implementiere ich einen `WindowListener`, der auf den `windowActivated`-Event des Fensters wartet und dann den Thread erstellt und startet.

Der Thread darf nicht im Konstruktor gestartet werden, da zu diesem Zeitpunkt noch nicht alle Ressourcen bereit sein könnten.

Wie viele Threads starten Sie?

Ich starte nur einen Thread.

Wie wird die Darstellung der Welt aktualisiert?

In der `run()`-Methode des Threads wird je nach Laufrichtung der Index der Bildliste hoch- oder runtergezählt. Das so ermittelte Bild wird der Zeichenfläche als Parameter übergeben, welche dann ihre Anzeige mittels `repaint()` und `paintComponent()` aktualisiert.

Wann sterben die Threads?

Der Thread kann nur sterben, wenn er von aussen unterbrochen wird. Das sollte grundsätzlich nie der Fall sein. Somit läuft der Thread so lange wie die Applikation läuft.

Was passiert, wenn die Applikation gestoppt wird?

Mittels `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);` wird das Programm beendet, wenn das Fenster geschlossen wird. Da der gestartete Thread eine Eigenschaft eines der GUI-Komponenten ist, wird dieser mit dem GUI zusammen "entsorgt".

3 Bankgeschäfte

Was sollte beim Szenario passieren, wenn das Programm korrekt ablaufen würde?

Nach den Überweisungen hätten alle Konti wieder das ursprüngliche Saldo.

Was beobachten Sie?

Es gibt Abweichungen: Vor und nach den Überweisungen weisen die Konti (in der Summe) unterschiedliche Saldi aus.

Wie erklären Sie sich das Programmverhalten?

Verschiedene Threads kommen sich in die Quere. Hier ein Beispiel:

- Thread A: überweist einen Betrag von Konto X auf Y
- Thread B: überweist einen Betrag von Konto Y auf X
- Konto X und Y haben den Anfangssaldo `1'000`
- Thread A arbeitet:
  - Das Saldo auf Konto X wird um `1` reduziert
  - Das Saldo von Konto Y soll um `1` erhöht werden
    - Das aktuelle Saldo von Konto Y wird als `1000` ausgelesen
- Thread B unterbricht:

    - Das Saldo auf Konto Y wird um 1 reduziert (Y: 999)
- Thread A unterbricht:
    - Das alte, als 1000 (statt 999) herausgelesene Saldo von Y wird um 1 erhöht
    - Saldo Y beträgt nun 1001
- Thread A ist fertig
- Thread B fährt fort:
    - Das Saldo auf Konto X wird um 1 erhöht (X: 1000)
- Thread B ist fertig
- Die Saldi sehen nun folgendermassen aus:
    - X: 1000 (korrekt)
    - Y: 1001 (falsch)

Es ist eine Geldeinheit aus dem Nichts entstanden!

Wie können Sie ein noch stärkeres Fehlverhalten provozieren?

Bei der Methode `deposite()` könnte der Thread zwischen dem Auslesen, dem Erhöhen und dem Zurückschreiben von `balance` etwas schlafengelegt werden, damit ein anderer Thread besser Gelegenheit hat, dazwischenzufunken:

```
public void deposite(final int amount) throws InterruptedException {
    int balance = this.balance;
    Thread.sleep(10);
    balance += amount;
    Thread.sleep(10);
    this.balance = balance;
}
```

Dadurch wird weiter unterstrichen, dass die Anweisung `this.balance += amount;` keineswegs atomar ist.

Tatsächlich wird die Abweichung so noch wesentlich grösser.

Welche Art von Synchronisation setzen Sie ein?

Ich synchronisiere nur jeweils die Instanz:

```
public void transfer(final BankAccount target, final int amount) {
    synchronized (this) {
        this.balance -= amount;
    }
    synchronized (target) {
        target.deposite(amount);
    }
}
```

Man könnte auch auf die Klasse synchronisieren:

```
public void transfer(final BankAccount target, final int amount) {
    synchronized (BankAccount.class) {
        this.balance -= amount;
        target.deposite(amount);
    }
}
```

Welche Art von Synchronisation ist für die Bankkonto-Klasse besser? Warum?

Es ist besser, Synchronisation auf die Instanzen anzuwenden. Synchronisierte man auf die ganze Klasse, müssten Überweisungen, die auf ganz verschiedenen Konti arbeiten, hintereinander ausgeführt werden, da zu jedem Zeitpunkt nur ein Thread auf ein Konto zugreifen dürfte. Die Anwendung würde so komplett serialisiert.

Was beobachten Sie nun?

Mit beiden Arten der Synchronisation funktioniert der Testfall nun korrekt. Es gibt aber grosse Performance-Unterschiede. Beispiel: Je `10'000` Konti überweisen sich `1'000.-` in Schritten von `1.-`hin- und zurück:

- Klassen-Synchronisation: `3.5-4.5` Sekunden
- Instanz-Synchronisation: `2.0-2.8` Sekunden

Wie erklären Sie sich das Programmverhalten?

Die Synchronisation verhindert, dass nicht aktuelle Werte für Berechnungen verwendet werden. Die Klassen-Synchronisation führt dazu, dass die Überweisungen ausschliesslich seriell ausgeführt werden, was die Performance-Einbusse erklärt.

Wie eliminieren Sie die Schwachstelle am besten?

mittels der Instanz-Synchronisierung (siehe oben)

Wie müssen Zugriffe auf gemeinsame Ressourcen am besten geschützt werden?

Es muss darauf geachtet werden, dass Lese-, Rechen- und Schreibanweisungen nicht unterbrechbar sind. Dies erreicht man entweder durch `synchronized`-Methoden oder `synchronized`-Blöcke.

Was sollte bei der Synchronisation in jedem Fall vermieden werden?

Die parallele Anwendung darf nicht durch unnötig weit gefasste Synchronisation zu einer seriellen Ausführung gezwungen werden, wie es mit der Klassen-Synchronisation (siehe oben) der Fall war.

Was verursacht der Einsatz von Synchronisation im Allgemeinen?

Durch die Synchronisation entstehen Stellen, an denen immer nur ein Thread aktiv sein kann. Dadurch ergeben sich Performance-Einbussen, die aber teils für einen korrekten Programmablauf in Kauf genommen werden müssen.

4 Das Ende eines Threads

Macht ein aktives Beenden von Threads überhaupt Sinn?

Das Beenden von Threads kann dann Sinn ergeben, wenn das Ergebnis der Berechnung eines Threads zwar erwünscht, aber nicht zwingend notwendig ist. Bei einem rechenintensiven 3D-Spiel könnte man so etwa auf eine rechenintensive Zusatzanimation verzichten, wenn deren Berechnung zu viel Zeit in Anspruch nimmt, um noch rechtzeitig dargestellt zu werden.

Bei einer Installations-Routine könnte ein Benutzer auch auf "Abbrechen" klicken, wodurch der Installations-Thread zwar beendet wird, die Applikation aber dennoch für Aufräumarbeiten weiterlaufen muss.

Welche Art des Abbruchs haben Sie implementiert?

Ich habe den Abbruch mit einem Attribut umgesetzt, da sich dies für eine `run()`-Methode, die mit einer Schleife arbeitet, einfach umsetzen lässt. Ausserdem war mit der Interrupt-Mechanismus nicht mehr geläufig. Beim Interrupt-Mechanismus müsste man kein eigenes Flag `stopped` verwenden.

Welche Art des Abbruchs finden Sie besser?

Ich finde den Abbruch mit einem Attribut für diesen Anwendungsfall besser, da hierbei im Gegensatz zum Interrupt-Mechanismus keine Exceptions behandelt werden müssen.

Was ist beim Abbruch mit Interrupt zu beachten?

Es muss auf die `InterruptedException` reagiert werden, indem man zumindest eine Log-Meldung oder die Stack-Trace ausgibt.

Warum ist die `stop()`-Methode `deprecated`?

Wird ein Thread mittels `stop()` angehalten, werden sämtliche von ihm gesperrten Datenobjekte in einem inkonsistenten Zustand wieder freigegeben, wodurch es zu Problemen kommen kann:

```
database.insert(monthlySalaryPayment);
// stop(): der Lohn wird als bezahlt markiert aber nicht überwiesen
monthlySalaryPayment.execute();
```

Darum sollte ein unterbrechbares Programm die Möglichkeit haben, auf die Unterbruchsanfrage reagieren zu können.

Die Methode könnte hilfreich sein, wenn der Benutzer eine längere Operation, die keine Seiteneffekte hat, gestartet und dann abgebrochen hat.

5 Optional: JoinAndSleep

In welchem Zustand ist ein Thread, der auf einen anderen Thread wartet?

Der Thread ist im Zustand “waiting”.

Welcher Programmteil muss die Threads abbrechen?

Die `main()`-Methode kann die Abbrechung anfordern, doch die `run()`-Methode des Threads muss der Aufforderung folgen.