Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
8. Januar 2018

10. Übung zur Vorlesung
Einführung in die Programmierung

A10-1 ***Nebenläufigkeit*** Erklären Sie die Antwort in jeweils 1–3 Sätzen:

a) Was ist der Unterschied zwischen Methode `run` und `start` bei Erben von `Thread`?

b) Was bedeutet das Schlüsselwort `synchronized` für eine Methodendeklaration?

c) Was bedeutet es, wenn im Rumpf einer gewöhnlichen (nicht statischen) Methode ein Block mit `synchronized(this){...}` vorkommt?
Z.B.: *`BlockA`*`; synchronized(this){ `*`BlockB`*` }; `*`BlockC`*`;`

d) Gegeben sei eine Klasse `A`, welche die synchronisierten Methoden `foo` und `bar` anbietet. Ein Thread ruft nun mit einem Objekt der Klasse `A` die Methode `foo` auf. Gleichzeitig möchte ein anderer Thread ebenfalls mit einem Objekt der Klasse `A` die Methode `bar` aufrufen. Was passiert, wenn es nur diese beiden Threads gibt?

e) Was ist der Unterschied zwischen einem wartenden und einem blockierten Thread?

f) Was ist der Unterschied zwischen einem Thread der die Methode `wait` aufruft und einem Thread welcher die Methode `sleep` aufruft?

g) Was ist der Unterschied zwischen einer Race Condition und einem Deadlock?

A10-2 ***Deadlocks*** Auf Vorlesungsfolie 12.45 wurde ein Beispiel vorgestellt, bei dem sich Deadlocks nicht mit `wait` und `notifyAll` verhindern lassen. Das Programm `Neben.java` finden Sie im Code zur Vorlesung zu Nebenläufigkeit.

Betrachten Sie eine Variation dieses Codes, welcher diesem Übungsblatt beiliegen sollte (Dateien `ConcurrentBanking.java` und `BankAccount.java`). Beschreiben Sie einen Programmablauf, bei dem ein Deadlock auftritt!

A10-3 ***Synchronisation*** Gegeben sind die drei Klassen `Main.java`, `Request.java` und `Sequence.java`, welche diesem Übungsblatt beiliegen sollten.

- Die Klasse `Sequence` verfügt über eine Klassenvariable `value`, und zwei statische Methoden `currentValue()` und `nextValue()`, die den aktuellen Wert `value` zurückgeben bzw. erhöhen.

  Die Deklaration der Klassenvariablen `value` als `volatile` stellt sicher, dass verschiedene Threads zum gleichen Zeitpunkt den gleichen Wert der Variable verwenden. Diese Art des synchronsierten Variablen-Zugriffs ist ein wenig schneller (und komfortabler), als alle Zugriffe über `synchronized`-Blöcke zu führen.

- `Request` erbt von `java.lang.Thread` und ruft in der `run()`-Methode nur die statische Methode `Sequence.nextValue()` auf, um einen eindeutigen Identifikator anzufordern.

- Die Klasse `Main` stellt den Einstiegspunkt des Programmes dar, und bekommt beim Aufruf mit übergeben, wie viele `Request`-Threads generiert werden sollen.

a) Rufen Sie `Main` mit verschiedenen Werten auf, z.B. `java Main 100` oder `java Main 1000`, so dass nicht alle `Request`-Threads eindeutige Identifikatoren zugewiesen bekommen, d.h. mindestens ein Identifikator (also ein Wert der Sequenz) mehrfach vergeben wird. Das Programm gibt eine entsprechende Meldung aus, falls dies passiert. Wiederholen Sie das Experiment mehrfach und erklären Sie das beobachtete Verhalten!

b) Ändern Sie die `Sequence`-Klasse so ab, dass ihre Methoden korrekt synchronisiert sind. In diesem Falle wird das Programm auch bei einer sehr großen Anzahl von `Request`-Threads keine mehrfach vergebenen Identifikatoren finden können.

H10-1 ***Nebenläufigkeit*** (8 Punkte; Abgabe: `H10-1.txt` oder `H10-1.pdf`)
Die Mitarbeiter des TCS Lehrstuhls trinken ständig Kaffee. Leider kann immer nur ein Mitarbeiter den Kaffeeautomaten bedienen: Der Mitarbeiter wählt ein Kaffee-Produkt aus, wartet bis es fertig ist, und entnimmt sein Produkt danach. Erst dann darf der nächste Mitarbeiter seinen nächsten Kaffee anfordern. Diesen Sachverhalt haben wir in Java mit den Klassen `CoffeeMachine` und `Benutzer` modelliert, welche diesem Übungsblatt beiliegen sollten.

a) Die Ausführung der `main`-Methode in `CoffeeMachine` sollte Ausgaben erzeugen, welche eine sinnvolle Benutzung des Kaffeeautomaten entsprechen.

   Leider ist dies aufgrund der Nebenläufigkeit jedoch nicht immer der Fall![1] Geben Sie den Anfang einer Ausgabe eines Programmablaufs an (2–4 Ausgabezeilen), welche einer illegalen Benutzung des Kaffeeautomaten entspricht, d.h. mehr als ein Benutzer benutzt den Kaffeeautomaten gleichzeitig. Erklären Sie auch kurz, wie es zu diesem Ablauf kam!

Für die restlichen Teilaufgaben b))–d)) betrachten wir eine veränderte Klasse `CoffeeMachine`, in der die Methoden `makeCoffee` und `takeCoffee` nun als `synchronized` deklariert sind.

b) Sind jetzt noch Programmabläufe möglich, bei der mehr als ein Benutzer gleichzeitig den Kaffeeautomaten verwendet? Begründen Sie Ihre Antwort kurz.

c) Im veränderten Programm kann es zu Deadlocks kommen. Erklären Sie warum!

d) Geben Sie genau an, wie die beiden Klassen `CoffeeMachine` und `Benutzer` durch Einfügen von Aufrufen der Methoden `wait()` und `notifyAll()` zu verändern sind, so dass jeder Programmablauf einer legalen Benutzung entspricht und keine Deadlocks mehr auftreten können!

Abgabe: Lösungen zu den Hausaufgaben können bis Sonntag, den 14.1.18, mit UniWorX nur als `.zip` abgegeben werden. Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (`www.tcs.ifi.lmu.de/lehre/ws-2017-18/eip/`).

[1] Verzahnung hängt von vielen Faktoren ab: Prozessortyp, Anzahl Kerne, OS, Auslastung, etc. Wir betrachten hier alle theoretisch möglichen Verzahnungen, nicht nur solche, welche Sie zufällig auf Ihrem eigenen Rechner beobachten können — Kunden mögen meistens keine konkrete Hardwarevorgaben!