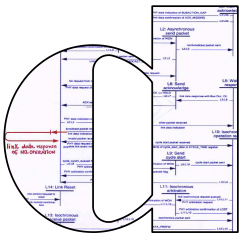
DEPENDABLE SYSTEMS AND SOFTWARE

Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Prof. Dr.-Ing. Holger Hermanns
Christian Eisentraut, M.Sc.



Nebenläufige Programmierung (Sommersemester '12)
Blatt P

Beachten Sie Folgendes: Wenn Sie Fragen zur Aufgabenstellung haben, benutzen Sie das *CMS-Forum* zur Diskussion. Zur Lösung bilden Sie bitte selbstständig eine Zweiergruppe mit einem Kommilitonen oder einer Kommilitonin. Senden Sie uns bis Freitag, den 06.07.2012, *pro Gruppe eine* Email an die Adresse `np-projekt@alan.cs.uni-sb.de`, deren Betreff die Matrikelnummer Ihrer beiden Gruppenmitglieder enthält. Der Betreff hat dabei das Format

Anmeldung, Fr. 06.07.2012, per Email

```
gruppe-<matrNrStudent1>-<matrNrStudent2>
```

Ihre Lösung der Aufgabe wird mit dem kompletten Quellcode und Dokumentation (siehe unten) in ein Archiv im `zip` Format zusammengefasst werden. Ihr Archiv für die Lösung hat den Namen `final-<matrNrStudent1>-<matrNrStudent2>.zip`. Ihre Email trägt den Betreff

```
final-<matrNrStudent1>-<matrNrStudent2>
```

und geht an die Adresse `np-projekt@alan.cs.uni-sb.de`. Spätmöglichster Abgabezeitpunkt ist Mittwoch, der 18.07.2012, um 23.59 Uhr MESZ.
Die Implementierungen werden von Ihnen in Form von kompilierenden *Sun Java 1.6* Code verfasst. Bei wiederholten Einsendungen gilt die zuletzt fristgerecht eingegangene Abgabe. Sie erhalten eine automatisch generierte Eingangsbestätigung für Ihre Email. Bei Problemen wenden Sie sich bitte an `eisentraut@cs.uni-sb.de`.

Meilenstein 2, Mi. 18.07.2012

Bis Sonntag, 08.07.2012, 23:59 Uhr MESZ, sollten Sie eine Dokumentation Ihres ersten Entwurf an `np-projekt@alan.cs.uni-sb.de` senden. Benennen Sie diesen Entwurf bitte

Meilenstein 1, So. 08.07.2012

```
entwurf-<matrNrStudent1>-<matrNrStudent2>.pdf
```

und den Betreff der Email `entwurf-<matrNrStudent1>-<matrNrStudent2>`. *Wichtig:* Details zum Meilenstein 1 finden Sie am Ende der Aufgabenstellung. Wir empfehlen eindringlich, Meilenstein 1 als Vorübung durchzuführen. Dies ist nicht verpflichtend. Jedoch: Wer Meilenstein 1 gewissenhaft bearbeitet *und* – bei Notwendigkeit – an dem daran anschließenden individuell zu vereinbarenden Besprechungstermin mit seinem Tutor teilnimmt, erhält die Chance, seine finale Abgabe bei Nicht-Bestehen nachzuarbeiten (ausgenommen, die Mängel der finalen Abgaben sind nicht mit einer gewissenhaften Bearbeitung der Aufgabenstellung vereinbar).

Wir werden wie gewohnt Office Hours abhalten um Ihre Fragen zu klären. Die Mittwochs-Übungen entfallen, wenn nicht anders angekündigt.

Aufgabe P.1
Im Folgenden sollen Sie eine typische Klausurkorrektur der Vorlesung *Nebenläufige Programmierung* als nebenläufiges Java-Programm umsetzen. Schauen Sie sich zunächst das im CMS bereitgestellte Code-Fragment an.
Die Klausuren werden von den Assistenten der Vorlesung korrigiert, während der Professor überprüft, die Endkorrektur vornimmt, und koordiniert. Der Professor und die Assistenten bilden dabei die Threads unseres Programms, während Klausuren Instanzen der Klasse `Exam` sind.
Die Assistenten sitzen alle an einem großen runden Tisch, in dessen Mitte der Professor thront, um alles gut im Blick zu haben.
Jeder Assistent korrigiert genau eine Aufgabe. Es gibt also genau so viele Aufgaben in der Klausur, wie es Assistenten gibt. Jeder Assistent hat einen Stoß Klausuren zu seiner Rechten und einen zu seiner Linken. Dabei ist wichtig, dass ein Stoß je von zwei Assistenten geteilt wird: der linke Stoß eines Assistenten ist der rechte Stoß seines linken Nachbars und umgekehrt. (Die Situation ist somit ähnlich wie bei den *Dining Philosophers*, nur dass wir Klausurstöße statt Gabeln zwischen den Assistenten liegen haben).
Wenn ein Assistent gerade keine Klausur korrigiert, so nimmt er sich eine Klausur von seinem rechten Stoß. Ist in dieser seine Aufgabe bereits korrigiert, so legt er die Klausur direkt auf seinen linken Stoß. Andernfalls korrigiert er zunächst seine Aufgabe in der Klausur und legt danach die Klausur auf seinen linken Stoß. Stellt er nach der Korrektur jedoch fest, dass die gesamte Klausur jetzt bereits korrigiert ist, so legt er sie auf den Klausurenstoß des Professors, damit dieser die Endkorrektur vornehmen kann.
Aufgaben korrigiert der Assistent, indem er die Methode `correct` der Klausur mit der entsprechenden Aufgabennummer aufruft. Diese Methode wird Ihren Prozessor ein wenig auslasten, um das Arbeiten der Assistenten zu simulieren.
Der Professor hat insgesant drei Aufgaben. Zum einen überprüft er fertig korrigierte Klausuren und nimmt danach die Endkorrektur vor. Dazu benutzt er die Methode `finish` der Klausur.
Zum anderen sorgt er dafür, dass die Assistenten immer möglichst gut beschäftigt sind, indem er gegebenenfalls Klausuren zwischen Stößen umverteilt.
Schließlich stellt der Professor auch fest, wann die Korrektur beendet ist, also alle Klausuren vollständig korrigiert wurden. Um zu erkennen, wann die Korrektur zu beenden ist, gibt es zwei Varianten:

(a) Im einfachen Fall kennt der Professor die Anzahl der Teilnehmer der Klausur, und damit der zu korrigierenden Klausuren.

(b) Im realistischen Fall kennt er (und seine Assistenten) diese Anzahl nicht, und muss prüfen, ob noch Klausuren im Umlauf sind.

Wenn der Professor die Korrektur beendet hat, schickt er die Assistenten nach Hause, worauf diese – ebenso wie der Professor – aufgrund übermäßiger Erschöpfung terminieren.
Vor Beginn der Korrektur werden die Klausuren möglichst gleichmäßig auf die Stöße verteilt. Dann beginnt die Arbeit. Das Aufteilen kann entweder durch den Professor oder seine Assistenten geschehen oder durch eine vor die Korrektur geschaltete Initialisierungsphase des Hauptprogramms. In dieser Phase darf die Gesamtzahl an Klausuren verwendet werden (auch bei Lösungsvariante (b) des Korrekturendes).

Ihre Aufgabe ist es nun, obiges Szenario in Java nebenläufig zu implementieren. Zum Bestehen des praktischen Projekts ist es notwendig, dass Ihre Lösung *funktional* korrekt ist[1], alle Speicherzugriffe *korrekt synchronisiert* sind und die Threads tatsächlich nebenläufig arbeiten[2]. Zudem muss das Programm ausreichend dokumentiert sein. Die Dokumentation umfasst eine sinnvolle Kommentierung des Quellcode sowie eine kurze Beschreibung Ihrer konkreten Implementierung (als extra pdf-Dokument oder innerhalb einer Quelldatei), die sich inhaltlich an den Leitfragen aus Meilenstein 1 orientiert.

- Meilenstein 1: Schreiben Sie ein detailliertes Dokument, in dem Sie einen Entwurf für Ihre Implementierung verständlich beschreiben und erklären. Dieses Dokument darf (muss aber nicht) bereits erste Code-Fragmente enthalten. Gehen Sie dabei auf folgende Leitfragen besonders ein:

[1] also der Aufgabenstellung entspricht
[2] Lösungen, bei denen die Thread-Ausführung *überwiegend* de facto sequentialisiert wird, können nicht bestehen

- Wie stellen Sie sicher, dass eine Klausur nie von mehreren Personen gleichzeitig bearbeitet wird?
- Welche abstrakte Datenstruktur (*Stack, Queue, Deque,...*) bietet sich zur Implementierung der Klausurstöße an? Bedenken Sie dabei folgende Gesichtspunkte: zwei Assistenten und der Professor greifen potentiell zur selben Zeit auf einen Stoß zu, jedoch mit verschiedenen Absichten. Wie können Sie hier korrekt synchronisieren und dabei alle Beteiligte möglichst wenig blockieren? Gegebenenfalls müssen Sie Ihre eigenen maßgeschneiderte Datenstruktur implementieren.
- Wie entscheidet der Professor, wann er welcher seiner drei Aufgaben nachkommen soll? Passen Sie dabei auf, dass der Professor nicht die meiste Zeit viel Prozessorleistung verlangt, ohne effektiv sinnvolle Arbeit zu tun, oder gar ständig die gesamte Korrektur lahmlegt. Finden Sie eine effektive Umverteilungsstrategie, die Assistenten möglichst selten arbeitslos werden lässt.
- Wie stellt der Professor das Korrekturende fest und wie teilt er dies den Assistenten mit?

Achtung: Sie können *auf eigenes Risiko* auf Meilenstein 1 verzichten. Wir werden jedoch nur Teilnehmern eine Chance auf Nacharbeit bei Nichtbestehen von Meilenstein 2 einräumen, die einen vollständig bearbeiteten Meilenstein 1 eingereicht haben und – bei Notwendigkeit – an dem daran anschließenden individuell zu vereinbarenden Besprechungstermin mit ihrem Tutor teilgenommen haben. Beachten Sie, dass das Bestehen des Praktischen Projekts (=Meilenstein 2) notwenige Voraussetzung zum Bestehen der Vorlesung ist.

- Meilenstein 2: Reichen Sie Ihre *gut dokumentierte* Implementierung bis Mittwoch, den 18.07.2012 ein (Details siehe erste Seite).

Bonus: Die Mitglieder der besten 10% aller Zweiergruppen erhalten einen Bonus von 0,3 auf ihre jeweilige Endnote. Um für den Bonus in Frage zu kommen, muss Variante (*b*) zum Erkennen des Korrekturendes implementiert werden.