Objektorientierte Programmiertechniken
LVA 185.A01, VU 2.0, 2011 W

8. Übungsaufgabe

Themen:

nebenläufige Programmierung, Exceptions

Termine:

Ausgabe:	14.12.2011
reguläre Abgabe:	21.12.2011, 13:45 Uhr
nachträgliche Abgabe:	11.01.2012, 13:45 Uhr

Abgabeverzeichnis:

Aufgabe8

Programmaufruf:

java Test

Grundlage:

Skriptum bis Seite 151, Schwerpunkt auf den Abschnitten 3.5 und 3.6

Aufgabe

Welche Aufgabe zu lösen ist:

Wie aus der einschlägigen Literatur [1] bekannt ist, entstand um das Jahr 50 v. Chr. in einem kleinen gallischen Dorf ein florierender Hinkelsteinmarkt, der bald auf das gesamte römische Reich überschwappte.

In der Vorweihnachtszeit will der findige Hinkelsteinerzeuger Verschenknix sein Unternehmen jedoch auf die Produktion von Christbäumen umstellen. Da ihm die Wichtigkeit reibungsloser Geschäftsprozesse bewusst ist, will er diese zuerst simulieren, und beauftragt Sie, ein entsprechendes Programm zu entwickeln:

Die Christbaumerzeugung besteht aus mehreren Produktionsschritten:

- Wildschweinjäger fangen im nahegelegenen Wald pro Jagd ein Wildschwein. Das gefangene Wildschwein legen sie in einem Kühlhaus ab.
- Köche verarbeiten je ein Wildschwein zu 5 Portionen Wildschweinbraten. Die fertigen Portionen werden auf einem Tisch angerichtet.
- Baumfäller nehmen eine Portion Wildschweinbraten zu sich, und fällen danach einen Christbaum aus dem nahegelegenen Wald. Der Christbaum wird in einem aufgelassenen Steinbruch gelagert.
- Logistiker verladen je 2 fertige Christbäume aus dem Steinbruch auf einen Ochsenkarren und fahren diesen auf den Dorfplatz. (Dort übernimmt ein eigenständiger Dienstleister den Ochsenkarren. Die Übernahme und die eigentliche Lieferung fällt in die Zuständigkeit des Dienstleisters und soll daher nicht implementiert werden.)

Das Unternehmen soll für 2 Wildschweinjäger, 1 Koch, 5 Baumfäller und 2 Logistiker simuliert werden.

Die verschiedenen Lager (Kühlhaus, Tisch, Steinbruch, Dorfplatz) können nur eine begrenzte Zahl an Produkten aufnehmen. Ist ein Lager voll, müssen die betroffenen Arbeiter so lange warten, bis im Lager wieder genug Platz ist, um etwas abzulegen.

Durch Konsum des berühmten Zaubertranks brauchen die Arbeiter keine Pausen; sie arbeiten in einem fort, wenn das notwendige Ausgangsmaterial vorhanden und das nachfolgende Lager frei ist.

Da die Zukunft des Christbaummarktes unsicher ist, gibt es die Möglichkeit der Unternehmensauflösung, bei der alle Arbeiter und Lager sofort ihre Arbeit beenden.

Simulieren sie das Unternehmen mittels eines nebenläufigen Java-Programms, wobei jeder Arbeiter durch je einen Thread dargestellt wird. Jeder Arbeiter benötigt eine gewisse Arbeitszeit für seinen Arbeitsschritt. Simulieren Sie diese Zeit mittels der Methode *Thread.sleep(n)* auf dem Thread, der den Arbeiter darstellt. Lassen Sie jeden Arbeiter einige Millisekunden arbeiten (für jeden Arbeiter unterschiedlich, z.B. zwischen 1 und 50 Millisekunden). Simulieren Sie die Unternehmensauflösung mittels eines Aufrufs der Methode *Thread.interrupt()* für jeden Thread. Dabei sollen alle Arbeiter (Threads) gestoppt und die Anzahl der bisher von jedem einzelnen Arbeiter abgelieferten Wildschweine, Wildschweinbraten, Christbäume und Ochsenkarren sowie die Lagerstände der einzelnen Lager auf der Standardausgabe angezeigt werden. Die Wildschweinjäger werden beim Start des Programms mit einer fixen Anzahl an Jagdausflügen beauftragt. Sobald alle Zwischenprodukte aller vorhergehenden Produktionsstufen verarbeitet wurden, sollen die Threads für die Arbeiter automatisch stoppen und (ebenso wie nach einer Unternehmensauflösung) die Anzahl der abgelieferten Produkte bzw. sich in den Lagern befindenden Wildschweine, Wildschweinbraten, Christbäume und Ochsenkarren ausgeben.

Die Klasse *Test* soll die wichtigsten Normal- und Grenzfälle überprüfen und die Ergebnisse in verständlicher Form in der Standardausgabe darstellen. Bitte achten Sie darauf, dass der Test nach kurzer Zeit von selbst terminiert – maximal nach 10 Sekunden. Zumindest die folgenden Überprüfungen sind durchzuführen:

- Testen Sie die Simulation mit mindestens drei unterschiedlichen Verarbeitungsläufen.
- Testen Sie die Simulation in den verschiedenen Verarbeitungläufen mit unterschiedlichen Werten für die Dauer der Arbeitsschritte der Arbeiter, und geben Sie dabei auch Arbeitern derselben Art unterschiedliche Arbeitsgeschwindigkeiten.
- Testen Sie auch mit verschiedenen Lagergrößen. Nehmen Sie so kleine Werte für die Lagergrößen an, dass zumindest bei einem Durchlauf Arbeiter warten müssen, bis Platz in einem Lager frei wird. Der Dorfplatz soll groß genug sein, um alle vorkommenden Karren aufnehmen zu können.

Warum die Aufgabe diese Form hat:

Das Unternehmen soll die nötige Synchronisation bildlich veranschaulichen und durch die Analogie zur realen Welt ein Gefühl für die vielen eventuell auftretenden Sonderfälle geben. Beispielsweise müssen die Arbeiter erkennen, wann sie nach erledigter Arbeit stoppen sollen. Einen speziellen Sonderfall stellt die Unternehmensauflösung dar, die jederzeit in jedem beliebigen Zustand auftreten kann. Dabei wird auch geübt, nach einer an einer beliebigen Programmstelle auftretenden Exception den Objektzustand so weit wie nötig zu rekonstruieren, um ein sinnvolles Ergebnis zurückliefern zu können.

Wie die Aufgabe zu lösen ist:

Überlegen Sie sich genau, wie und wo Sie Synchronisation verwenden. Halten Sie die Granularität der Synchronisation möglichst klein, um unnötige Beeinflussungen anderer Threads zu reduzieren. Vermeiden Sie aktives Warten, indem Sie immer *sleep* aufrufen, wenn Sie eine bestimmte Zeit warten müssen. Beachten Sie dabei auch, dass ein Aufruf von *sleep* innerhalb einer *synchronized*-Methode oder *synchronized*-Anweisung den entsprechenden Lock nicht freigibt. Für die Simulation eines Arbeiters ist das erwünscht, da der Arbeiter während eines Arbeitsschritts nichts anderes machen können soll. An anderen Programmstellen ist ein derartiges Verhalten aber unerwünscht.

Testen Sie Ihre Lösung bitte rechtzeitig auf der g0, da es im Zusammenhang mit Nebenläufigkeit große Unterschiede zwischen den einzelnen Plattformen geben kann. Ein Programm, das auf einem Rechner problemlos funktioniert, kann auf einem anderen Rechner (durch winzige Unterschiede im zeitlichen Ablauf) plötzlich nicht mehr funktionieren. Dieser Indeterminismus macht das

Testen nebenläufiger Programme äußerst schwierig.

Nebenläufigkeit kann die Komplexität eines Programms gewaltig erhöhen. Achten Sie daher besonders darauf, dass Sie den Programm-Code so klein und einfach wie möglich halten. Jede unnötige Anweisung kann durch zusätzliche Synchronisation (oder auch fehlende Synchronisation) eine versteckte Fehlerquelle darstellen und den Aufwand für die Fehlersuche um vieles stärker beeinflussen als in einem sequentiellen Programm.

Was im Hinblick auf die Beurteilung zu beachten ist:

Der Schwerpunkt bei der Beurteilung liegt auf korrekter nebenläufiger Programmierung und der richtigen Verwendung von Synchronisation sowie dem damit in Zusammenhang stehenden korrekten Umgang mit Exceptions. Kräftige Punkteabzüge gibt es für

- fehlende oder fehlerhafte Synchronisation,
- zu große Synchronisationsbereiche, durch die sich Threads gegenseitig unnötig behindern,
- nicht richtig abgefangene Exceptions im Zusammenhang mit nebenläufiger Programmerierung,
- Nichttermination von *java Test* nach 10 Sekunden,
- unnötigen Code und mehrfache Vorkommen gleicher oder ähnlicher Code-Stücke
- vermeidbare Warnungen des Compilers, die mit (der falschen Verwendung von) Generizität in Zusammenhang stehen,
- Verletzungen des Ersetzbarkeitsprinzips bei Verwendung von Vererbungsbeziehungen,
- mangelhafte Zusicherungen,
- schlecht gewählte Sichtbarkeit,
- unzureichendes Testen
- und mangelhafte Funktionalität des Programms.

Was im Hinblick auf die Abgabe zu beachten ist:

Verzichten Sie wie üblich auf die Verwendung von packages und Verzeichnissen innerhalb des Abgabeverzeichnisses. Geben Sie (abgesehen von geschachtelten Klassen) nicht mehr als eine Klasse in jede Datei, und verwenden Sie aussagekräftige Namen. Achten Sie darauf, dass Sie keine Java-Dateien abgeben, die nicht zu Ihrer Lösung gehören.

Literatur

[1] RenÃ© Goscinny und Albert Uderzo, “Obelix GmbH & Co. KG”. In: *Asterix*, Band 23. Ehapa, 1978.