Objektorientierte Programmiertechniken
LVA 185.A01, VU, 3 Ects, 2012/2013 W

7. Übungsaufgabe

Themen:

Exceptions, nebenläufige Programmierung

Termine:

Ausgabe:	28.11.2012
reguläre Abgabe:	05.12.2012, 12:00 Uhr
nachträgliche Abgabe:	12.12.2012, 12:00 Uhr

Abgabeverzeichnis:

Gruppe/Aufgabe7

Programmaufruf:

java Test

Grundlage:

Skriptum, Schwerpunkt auf den Abschnitten 4.1 und 4.2

Aufgabe

Welche Aufgabe zu lösen ist:

Nachdem Sebastian Vettel wieder Weltmeister ist, in der Winterpause keine Rennen stattfinden und er daher auf Autodrom umsteigt, der Wiener Prater aber zu teuer ist, soll eine Simulation für ein Autodrom entwickelt werden. Dazu wird die rechteckige Fahrbahn schachbrettartig in viele einzelne Felder unterteilt. Ein Auto befindet sich auf einem Feld und ist nach einer der vier Himmelsrichtungen (Norden, Osten, Süden und Westen) ausgerichtet. Das Fahren wird in viele unterschiedlich lange dauernde Teilschritte unterteilt. In jedem Schritt kann sich ein Auto vom aktuellen Feld auf ein benachbartes Feld bewegen. Es gibt zwei Arten von Autos, ein schnelleres und ein beweglicheres. Von den acht benachbarten Feldern kann sich das schnellere Auto auf das nächste in Fahrtrichtung liegende Feld (3, siehe das 3x3-Feld unten) nach vorne

bewegen, auf das Feld schräg links nach vorne (2) bei gleichzeitiger Änderung der Fahrtrichtung um 90° nach links oder auf das Feld schräg rechts nach vorne (4) bei gleichzeitiger Änderung der Fahrtrichtung um 90° nach rechts. Das beweglichere Auto kann sich zusätzlich noch auf das Feld links (1) bei gleichzeitiger Änderung der Fahrtrichtung um 90° nach links und auf das Feld rechts (5) bei gleichzeitiger Änderung der Fahrtrichtung um 90° nach rechts bewegen. Rückwärtsfahren (Felder (6), (7) und (8)) ist nicht möglich.

```
8 1 2
7 > 3
6 5 4
```

Befinden sich bereits Autos auf dem Feld, so gibt es einen Zusammenstoß. Für jedes Auto, das man von vorne trifft, gibt es einen Bonuspunkt, wird man von einem anderen Auto hinten, links oder rechts getroffen, so gibt es einen Minuspunkt. Ziel ist es möglichst viele Punkte zu sammeln. Die Autos bewegen sich mit unterschiedlichen möglichst einfachen Strategien weiter, z.B. zufällig, oder im Kreis oder in Schlangenlinien im Kreis, aber nie über die Fahrbahn hinaus.

Simulieren sie das Autodrom mittels eines nebenläufigen Java-Programms. Stellen sie dabei jedes Auto durch einen eigenen Thread dar. Jedes Auto bewegt nach einer gewissen Zeit (wenige Millisekunden) von einem Feld zu einem Nachbarfeld, ein schnelles Auto nach sehr wenigen Millisekunden, ein bewegliches Auto nach etwas mehr Millisekunden. Simulieren Sie Wartezeiten mittels der Methode *Thread.sleep(n)*. Achtung: *sleep* behält alle Monitore (= Locks); Sie sollten *sleep* daher nicht innerhalb einer *synchronized*-Methode oder -Anweisung aufrufen, wenn während der Wartezeit von anderen Threads aus auf dasselbe Objekt zugegriffen werden soll. Wenn ein Auto die Maximalpunktezahl (10) erreicht hat, oder ein Auto eine Maximalzahl an Feldwechseln ausgeführt hat, geben Sie von allen Autos die Punktezahl aus und beenden alle Threads. Verwenden Sie *Thread.interrupt()* um einen Thread zu unterbrechen, geben Sie den Namen und den Punktestand aus, und beenden Sie den Thread. Implementieren Sie zumindest zwei unterschiedliche Bewegungsstrategien.

Die Klasse “Test” soll (nicht interaktiv) Testläufe des Autodroms durchführen und die Ergebnisse in allgemein verständlicher Form in der Standardausgabe darstellen. Bitte achten Sie darauf, dass die Testläufe nach kurzer Zeit terminieren (maximal 10 Sekunden für alle zusammen). Führen Sie mindestens drei Testläufe mit unterschiedlichen Einstellungen durch:

- Jeder Testlauf soll eine unterschiedliche Menge an Autos mit unterschiedlicher Beweglichkeit und Bewegungsstrategie vor dem Start auf unterschiedliche Felder der Fahrbahn positionieren. Auch innerhalb eines Testlaufs sollen Autos unterschiedliche Geschwindigkeiten haben.
- Stellen Sie die Parameter so ein, dass irgendwann Autos auch zusammenstoßen.

Warum die Aufgabe diese Form hat:

Die Simulation soll die nötige Synchronisation bildlich veranschaulichen und ein Gefühl für eventuell auftretende Sonderfälle geben. Beispielsweise müssen Autos erkennen, wenn sie die maximale Punkteanzahl erreicht haben. Einen speziellen Sonderfall stellt das Simulationsende dar, das (aus Sicht eines Autos) jederzeit in jedem beliebigen Zustand auftreten kann. Dabei wird auch geübt, nach einer an einer beliebigen Programmstelle aufgetretenen 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, dass ein Aufruf von *sleep* innerhalb einer synchronized-Methode oder -Anweisung den entsprechenden Lock nicht freigibt.

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.

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 Programmerierung und der richtigen Verwendung von Synchronisation sowie dem damit in Zusammenhang stehenden korrekten Umgang mit Exceptions. Punkteabzüge gibt es für

- fehlende oder fehlerhafte Synchronisation,
- zu große Synchronisationsbereiche, durch die sich Threads gegenseitig unnötig behindern (z.B. die gesamte Fahrbahn als Synchronisationsobjekt),
- nicht richtig abgefangene Exceptions im Zusammenhang mit nebenläufiger Programmerierung,
- Nichttermination von *java Test* innerhalb von 10 Sekunden,

- unnötigen Code und mehrfache Vorkommen gleicher oder ähnlicher Code-Stücke,
- vermeidbare Warnungen des Compilers, die mit 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.