Universität Stuttgart

Institut für Visualisierung
und Interaktive Systeme

Übungsblatt 11

Datenstrukturen und Algorithmen (SS 2016)

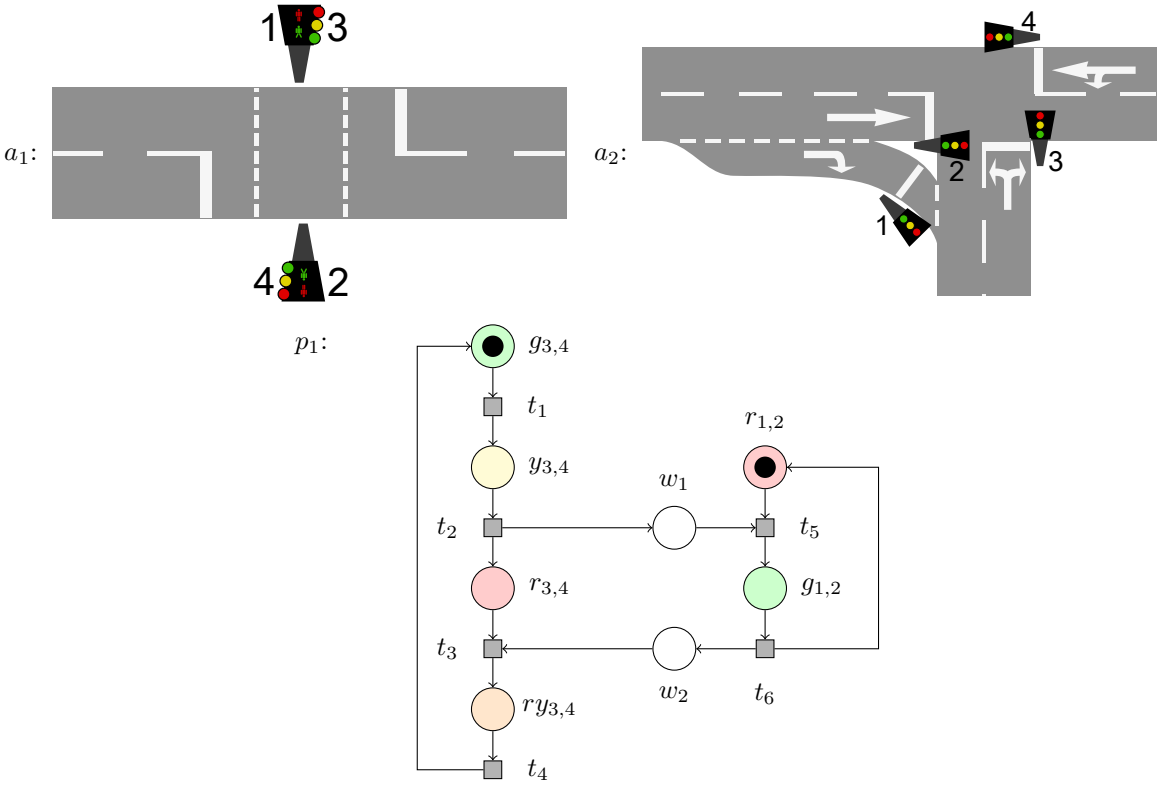
Abgabe: Mittwoch, 07.07.2016, 23:59 Uhr — Besprechung: ab Montag, 11.07.2016

Bitte lösen Sie das Übungsblatt in **Gruppen von 3 Studenten** und wählen EINEN Studenten aus, welcher die Lösung im ILIAS als **PDF** als **Gruppenabgabe** (unter Angabe aller Gruppenmitglieder) einstellt. Bitte erstellen Sie dazu ein **Titelblatt**, welches die Namen der Studenten, die Matrikelnummern, und die E-Mail-Adressen enthält.

Dieses Übungsblatt beinhaltet 3 Aufgaben mit einer Gesamtzahl von 30 Punkten.

Aufgabe 1 Ampelschaltung *[Punkte: 14]*

Gegeben seien die folgenden Ampelanlagen a_1 und a_2 sowie das (zur Verdeutlichung eingefärbte) Petrinetz p_1, das a_1 modelliert. Die Zustände, beziehungsweise Stellen einer Ampel X, seien als r_X = rot, ry_X = rot-gelb, g_X = grün und y_X = gelb definiert. Mehrere Ampeln X und Y im selben Zustand z können als $z_{X,Y}$ zusammengefasst werden.



Gehen Sie jeweils davon aus, dass Transitionen stets nach einer gewissen Wartezeit geschaltet werden, die hier nicht abgebildet ist. Das heißt, jede Marke bleibt $> 0s$ in jeder Stelle. Jede Stelle des Petrinetz hat eine Kapazität von Eins.

(a) *(2 Punkte)* Die Schaltung der Ampelanlage a_1 wird durch das Petrinetz p_1 dargestellt. Wozu dienen die Zustände w_1 und w_2? Erläutern Sie deren Nutzen in 2–3 Sätzen.

(b) *(3 Punkte)* Erweitern Sie p_1 dahingehend, dass die Fußgängerampel in eine reine Bedarfsampel, die nur beim Druck auf einen Knopf grün wird, verwandelt wird. Achten Sie darauf, keine Fußgänger zu gefährden.

(c) *(4 Punkte)* Eine einzelne Ampel lässt sich wie der linke Teil von p_1 darstellen (also der Teilgraph bestehend aus $g_{3,4}$, t_1, $y_{3,4}$, t_2, $r_{3,4}$, t_3, $ry_{3,4}$ und t_4). Dabei stellt jede Stelle einen logischen Zustand der Ampel – grün, gelb, rot, rot-gelb – dar.

Stellen Sie nun eine einzelne Ampel mit nur drei Stellen r, y und g dar, von denen jede den Zustand eines der Lichter der Ampel repräsentiert – der Zustand rot-gelb soll entsprechend durch zwei der drei Stellen repräsentiert werden. Es dürfen nur gültige Ampelzustände aktiviert werden, und diese nur in der richtigen Reihenfolge. Geben Sie, wenn nötig, Kapazitäten für die Stellen an.

(d) *(5 Punkte)* Stellen Sie ein Petrinetz für eine Ampelschaltung auf, welche a_2 steuert. Sorgen Sie dafür, dass jede Ampel so lange wie möglich im Zustand *grün* ist, dass allerdings auch niemals zwei Ampeln gleichzeitig grün sind, wenn sich dadurch überkreuzende Fahrzeugströme ergeben würden.

Aufgabe 2 Das Raucherproblem *[Punkte: 7]*

Ein Raucher benötigt zur Befriedigung seiner Sucht drei Dinge – Filter, Paper und Tabak. Stellen Sie sich folgendes Szenario vor:

Drei Kettenraucher sitzen an einem Tisch. Jeder der drei Raucher hat dauerhaft das Bedürfnis zu rauchen. Daher übt jeder Raucher auch nur eine von zwei Tätigkeiten aus – entweder er raucht oder er wartet darauf, dass er endlich wieder rauchen kann. Jeder Raucher hat eine andere der drei Zutaten Filter, Paper oder Tabak dabei – von dieser jedoch einen unendlichen Vorrat.

Zu Beginn des Szenarios werden daher ein Filter, ein Paper und eine Portion Tabak auf den Tisch gelegt. Jeder Raucher bemüht sich augenblicklich darum, die beiden Zutaten, die ihm fehlen, in seinen Besitz zu bekommen. Verfügt einer von ihnen über alle Zutaten, so kann er endlich rauchen gehen. Da Raucher natürlich soziale Menschen sind, besorgt er, nachdem er seine Sucht befriedigt hat, die beiden entwendeten Zutaten und legt beides zurück auf den Tisch.

Sie erkennen natürlich sofort das Problem, das sich bei dieser Wettlaufsituation ergibt. Es ist nicht sichergestellt, dass das System nicht verklemmt, d. h. jeder Raucher hat sich eine Zutat vom Tisch genommen und wartet auf die zweite.

(a) *(3 Punkte)* Modellieren Sie das Raucherproblem als Petrinetz. Machen Sie deutlich, wofür die einzelnen Zustände und Transitionen stehen.

(b) *(1 Punkt)* Einer Ihrer Kommilitonen schlägt zur Lösung dieses Problems vor, *Semaphore* einzusetzen. In dem im ILIAS bereitgestellten Projekt finden Sie die Klassen `Raucher`, `Simulation` und `Tisch`, in denen diese Lösung realisiert wurde. Wenn Sie die Simulation ausführen, stellen Sie fest, dass es trotz Einsatz der Semaphore zu Verklemmungen kommen kann.

Beschreiben Sie, welchen Fehler Ihr Kommilitone bei seinem Lösungsansatz gemacht hat.

(c) *(1 Punkt)* Beschreiben Sie, wie Sie den Lösungsansatz Ihres Kommilitonen verbessern würden. Auch weiterhin soll das Raucherproblem mittels Semaphore gelöst werden, aber es soll zu keinen Verklemmungen mehr kommen können.

(d) *(2 Punkte)* Neben Semaphoren gibt es noch weitere Konzepte zur Synchronisation von nebenläufigen Prozessen. Beschreiben Sie, wie die Klasse `Tisch` angepasst werden müsste, damit diese keine Semaphore mehr verwendet, sondern die Synchronisation mittels Monitoren realisiert.

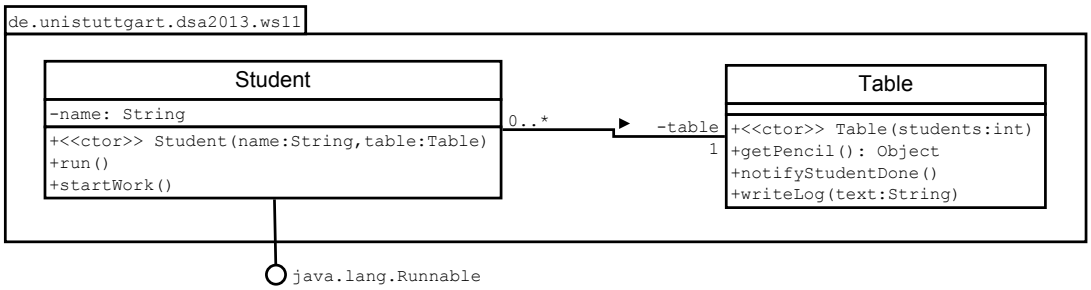
Sie können diese Aufgabe rein textuell lösen oder Java-Code-Fragmente angeben.

Aufgabe 3 Multithreading in Java *[Punkte: 9]*

Vorgeschichte: *Mehrere Studenten sitzen um einen Tisch und arbeiten am Übungsblatt. Leider haben sie gemeinsam nur einen einzigen Bleistift, der in der Mitte des Tisches liegt. Jeder Student denkt dabei eine Weile nach, nimmt dann den Bleistift, sobald dieser frei ist, schreibt an seiner Lösung und legt den Bleistift wieder zurück.*

Zusammen mit diesem Übungsblatt erhalten Sie einen Tisch mit einem Bleistift (virtuell, als Java-Code). Zusätzlich erhalten Sie ein Testprogramm, in dem einige Studenten gemeinsam an einem Tisch arbeiten.

(a) *(6 Punkte)* Programmieren Sie eine Klasse `Student` gemäß dem folgenden UML-Klassendiagramm:



`run()` soll dabei das Verhalten des Studenten nachbilden. Der Student soll mehrfach eine zufällig bestimmte Dauer nachdenken (`Thread.sleep`), dann versuchen, den Bleistift zu nehmen, eine Weile etwas schreiben (wiederum `Thread.sleep`) und den Stift zurücklegen. Das Nehmen und Zurücklegen des Stifts muss durch einen `synchronized`-Block ausgedrückt werden.

Verwenden Sie die `writeLog`-Methode des Tischs, um wichtige Ereignisse festzuhalten. Diese müssen mindestens den Versuch, den Stift zu nehmen, das tatsächliche Aufnehmen des Stifts und das Zurücklegen des Stifts umfassen. Achten Sie darauf, in den Protokollnachrichten den Namen des Studenten zu erwähnen.

Ist ein Student mit der Arbeit fertig, muss die `notifyStudentDone`-Methode des Tischs aufgerufen werden.

Implementieren Sie `Student.startWork()` so, dass ein neuer Thread für den Studenten gestartet wird.

Versehen Sie Ihren Code mit Dokumentationskommentaren und werfen Sie geeignete Exceptions, wenn ungültige Parameterwerte übergeben werden.

Die Abgabe soll aus der Java-Quelltextdatei `Student.java` bestehen.

(b) *(3 Punkte)* Stellen Sie den Ablauf am Beispiel von drei Studenten in Form eines Petrinetzes dar. Darin müssen die Zustände der Studenten (Denken und Schreiben) sowie die Verfügbarkeit des Bleistifts erkennbar sein.