Institut für Informatik
Prof. Dr.-Ing. Elke Pulvermüller
Lars Huning, M.Sc.

Universität Osnabrück, 14.06.2021
Testat bis 23.06.2021, 18:00 Uhr

Übungen zu Einführung in die Software-Entwicklung

Sommersemester 2021

Blatt 9

Aufgabe 9.1: Monitore und Locks (30 Punkte)

Betrachten Sie die Klassen `util.Queue`, `threads.Sleeper`, `threads.RandomGenerator` und `threads.MakeRandomSleeps`. Beim Ausführen der `main`-Methode von `threads.MakeRandomSleeps` kommt es zu zahlreichen Fehlern, da u.a. die verwendete Implementation des ADT Queue `util.Queue` nur eine vorgegebene Anzahl von Objekten aufnehmen kann.

Beheben Sie die Probleme dadurch, dass Sie dafür sorgen das:

- beim Löschen von Elementen aus einer leeren Queue solange gewartet wird, bis die Queue nicht mehr leer ist.
- beim Einfügen von Elementen in eine volle Queue solange gewartet wird, bis die Queue nicht mehr voll ist.

Nutzen Sie dafür zwei verschiedene Ansätze, die sie unabhängig voneinander implementieren.

1. Verändern Sie nur die Klassen `threads.RandomGenerator` und `threads.Sleeper` und beheben Sie die Probleme durch die Synchronisation über ein geeignetes Monitor-Objekt.
2. Verändern Sie nur die Klasse `util.Queue` und beheben Sie die Probleme, indem Sie möglichst kleine Blöcke durch `java.util.concurrent.locks.Lock`-Instanzen sperren. Nutzen Sie eine oder mehrere `java.util.concurrent.locks.Condition`-Instanzen um die verlangte Synchronisation zu gewährleisten.

Aufgabe 9.2: Nebenläufiges Ameisenrennen (35 Punkte)

Vervollständigen Sie das Programm `AntRace`, das die Länge aller kürzesten Wege von einem Startfeld zu allen anderen Feldern auf einem Spielfeld berechnet (*all-pairs-shortest-path*).

Die Klassen `antRace.Field`, `antRace.AntRace`, `antRace.Ant` und das Interface `antRace.Fields` sind vorgegeben und können von Ihnen nach Belieben verändert und erweitert werden. Sie können die Anfangsparameter ihres Programms in der `main`-Methode hart codieren, d.h. Sie brauchen keine Benutzereingabe zu implementieren.

Arbeiten Sie zur Berechnung der kürzesten Wege mit eigenständigen Threads. Jeder Ihrer Threads sei eine Ameise. Die Klasse `antRace.Ant` gibt ein `Runnable` mit einem Konstruktor für die

Ameisen-Threads vor. Eine neue Ameise wird mit einem Startfeld und der bisher gelaufenen Anzahl an Schritten erzeugt. Die erste Ameise würde also mit dem Startfeld und `1` aufgerufen. Bei der Erzeugung setzt jede Ameise die Schrittzahl auf Ihrem Startfeld auf ihre eigene Schrittzahl. Danach schaut sich eine Ameise in jedem Durchlauf alle 8 Nachbarfelder (Moore-Nachbarschaft) an. Ist ein Feld als frei gekennzeichnet oder die dort eingetragene Schrittzahl größer als die Schrittzahl der Ameise plus eins, hat die Ameise einen neuen Weg gefunden. Den ersten neuen Weg beschreitet die Ameise selbst, d.h. sie erhöht Ihre eigene Schrittzahl um eins und verändert ihr Feld auf das Nachbarfeld. Für jeden weiteren Weg, den die Ameise im aktuellen Durchlauf findet, erzeugt sie eine neue Ameise mit dem Nachbarfeld und der um eins erhöhten Schrittzahl. Eine Ameise ist fertig, wenn sie in ihrer Nachbarschaft keine neuen Wege mehr findet.

Wenn alle Ameisen Ihre Arbeit beendet haben, steht in jedem Feld, das vorher auf `0` gesetzt war, eine natürliche Zahl ≥ 1, die die Länge des kürzesten Weges zum Startfeld darstellt. Den kürzesten Weg von einem Feld zum Startfeld erhielte man also, indem man solange zur nächst kleineren Zahl wandert, bis man beim Startfeld angekommen ist.

Achten Sie darauf, dass Ihr Programm erst dann beendet wird und die Länge der kürzesten Wege pro Feld auf der Standardkonsole ausgibt, wenn die letzte Ameise ihre Arbeit beendet hat. Ein `Thread` kann auf die Beendigung eines anderen `Thread` warten, in dem er an dem anderen `Thread` die Methode `join` aufruft.

Aufgabe 9.3: Fragen (35 Punkte)

Beantworten Sie Ihrer Tutorin / Ihrem Tutor Fragen zu den Themen dirty read und lost update, sowie allgemeine Fragen zum Thema Synchronisation.