PFP-Team
pfp@i2.cs.fau.de

Übungsblatt 4

Abgabe bis 29.05.2019, 18:00 Uhr

Aufgabe 4.1: Thread-Sicherheit

a) Wann sind Klassen korrekt? Wie kann man nicht-korrekte Klassen erkennen?

b) Wann sind Klassen thread-sicher? Wie kann man nicht-korrekte parallele Klassen erkennen?

c) Nehmen Sie zu folgender Aussage Stellung: „Ein paralleles Programm, das nur thread-sichere Klassen verwendet und bei sequentieller Ausführung korrekt arbeitet, ist thread-sicher!“

Aufgabe 4.2: Sichtbarkeiten

a) Aus welchen Gründen ist Sichtbarkeitssynchronisation bei parallelen Programmen wichtig?

b) Welche Rollen spielen hierbei das Schlüsselwort `volatile` und das Paket `java.util.concurrent.atomic`?

c) Welche Sichtbarkeitsprobleme treten in folgendem Programm auf? Wie kann man diese durch den Einsatz von `synchronized` und/oder `volatile` lösen?

```
public class Unsichtbarer {

    private String name;
    private int alter;
    private double gewicht;

    public Unsichtbarer(String name, int alter, double gewicht) {
        this.name = name;
        this.alter = alter;
        this.gewicht = gewicht;
    }

    public void umtaufen(String name) { this.name = name; }

    public String getName() { return name; }

    public int getAlter() { return alter; }

    public void geburtstag() { alter++; }

    public double waage() { return gewicht; }

    public void sport(double gewicht) {
        this.gewicht = gewicht;
    }
}
```

Aufgabe 4.3: Fehlerquellen

a) Welche neuen Fehlerquellen sind bei der Programmierung mit mehreren Aktivitätsfäden vorhanden, die bei der Verwendung von nur einer Aktivität nicht auftreten?

b) Welche Gegenmaßnahmen und Programmiermittel fallen Ihnen dazu ein?

Bonusaufgabe 4.4: Notfall-Protokollierung — 3 Punkte

Um die Wartezeit auf Hilfe bei einem Notfall zu minimieren, möchte eine Notrufstelle herausfinden, in welchen Regionen welche Notfälle besonders häufig auftreten. Helfen Sie dabei, indem Sie eine Klasse `AccidentLoggerImpl` erstellen, die das Interface `AccidentLogger` implementiert und parallel zählt, wie oft ein bestimmter Notfall in einer bestimmten Region vorkommt.
Teilen Sie das übergebene `Accident`-Array, welches alle gemeldeten Notfälle enthält, möglichst gleichmäßig auf `nThreads` Arbeiterthreads auf. Die Threads sollen **ohne** eigene blockierende Synchronisation (`synchronized`) ermitteln, wie häufig jede Kombination von Notfallart und Notfallort im Array vorkommt. Machen Sie sich dazu mit der Klasse `ConcurrentHashMap` aus dem Paket `java.util.concurrent` vertraut, die eine thread-sichere Implementierung einer Hash-Tabelle darstellt. Erstellen Sie *eine* Instanz einer `ConcurrentHashMap` und füllen Sie diese mit allen Threads parallel, indem Sie als Schlüssel die Notfälle und als Wert die bisherige Anzahl verwenden. Speichern Sie nach erfolgter Berechnung jede aufgetretene Kombination aus Notfallart und Notfallort mit der Anzahl der Vorkommen in einem `AccidentWithCount`-Objekt und geben Sie diese als Array zurück. Die Reihenfolge der `AccidentWithCount`-Objekte im Ergebnisarray ist unerheblich. Testen Sie Ihre Lösung mit dem zur Verfügung gestellten `AccidentLoggerTest`. Achten Sie außerdem darauf, dass Ihre Lösung beliebige Notfallarten und -orte verarbeiten kann.

Bonusaufgabe 4.5: Sichtbarkeit und Wettlaufsituationen — 3 Punkte

In dieser Aufgabe sollen Sie die Ausgabe zweier paralleler Programme (`Sync1` bzw. `Sync2` – auch online auf der Homepage verfügbar) analysieren. Nehmen Sie dabei an, dass die Ausführung unmittelbar in den jeweiligen `main`-Methoden beginnt. Geben Sie Ihre Lösung in der Datei **pfp_abgabe_4.pdf** im EST ab.

a) Ist sichergestellt, dass `Sync1` bei jeder Ausführung terminiert? Begründen Sie Ihre Antwort.

b) Welche Ausgaben können die terminierenden Ausführungen von `Sync1` produzieren? Begründen Sie Ihre Antwort.

c) Ist sichergestellt, dass `Sync2` bei jeder Ausführung terminiert? Begründen Sie Ihre Antwort.

d) Welche Ausgaben können die terminierenden Ausführungen von `Sync2` produzieren? Begründen Sie Ihre Antwort.

```java
import java.util.concurrent.*;

public class Sync1 {
    static boolean finished = false;
    static int result = 0;

    static class TaskA implements Runnable {
        int id;

        TaskA(int id) {
            this.id = id;
        }

        public void run() {
            if (id == 0) {
                result = 1;
                finished = true;
            } else {
                while (!finished) { /* do nothing */ }
                result = 2;
            }
        }
    }

    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newFixedThreadPool(4);
        executor.execute(new TaskA(0));
        executor.execute(new TaskA(1));
        executor.shutdown();
        while (!executor.awaitTermination(1, TimeUnit.SECONDS)) { }
        System.out.println(result);
    }
}
```

```java
public class Sync2 {
    static class TaskB extends Thread {
        int input;
        int result;
        volatile boolean finished = false;

        TaskB(int input) {
            this.input = input;
        }

        public void run() {
            if (input < 2) {
                result = input;
            } else {
                TaskB t1 = new TaskB(input - 1);
                TaskB t2 = new TaskB(input - 2);
                t1.start();
                t2.start();
                while (!t1.finished || !t2.finished) { }
                result = t1.result + t2.result;
            }
            finished = true;
        }
    }

    public static void main(String[] args) throws Exception {
        TaskB task = new TaskB(6);
        task.start();
        while (!task.finished) { }
        System.out.println(task.result);
    }
}
```

6 Punkte