

Übungsblatt 4

Abgabe bis 16.05.2018, 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; }

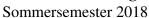
public int getAlter() { return alter; }

public void geburtstag() { alter++; }

public double waage() { return gewicht; }

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

Parallele und funktionale Programmierung



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



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: GradeCounter

3 Punkte

Durch die steigenden Studierendenzahlen wird es immer schwieriger, die Notenstatistiken für die AuD- und PFP-Klausuren zu erzeugen. Helfen Sie dabei, indem Sie eine Klasse GradeCounterImpl erstellen, die das Interface GradeCounter implementiert und parallel zählt, wie oft jede Note vorkommt.

Teilen Sie das übergebene String-Array, welches alle vorkommenden Noten enthält, möglichst gleichmäßig auf nThreads Arbeiterthreads auf. Die Threads sollen ohne eigene blockierende Synchronisation (synchronized) ermitteln, wie häufig eine Note 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 befüllen Sie diese mit allen Threads parallel, indem Sie als Schlüssel die Noten und als Wert die bisherige Anzahl verwenden.

Erstellen Sie für jede vorkommende Note ein GradeCount-Objekt, sobald alle Threads fertig sind, und geben Sie diese als Array zurück. Ein GradeCount-Objekt enthält die Note und deren Anzahl. Die Reihenfolge der GradeCount-Objekte im Ergebnisarray ist unerheblich. Testen Sie Ihre Lösung mit dem zur Verfügung gestellten GradeCounterTest.

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 zu Beginn alle Attribute die initial im Code zugewiesenen Werte enthalten. Geben Sie Ihre Lösung in der Datei **pfp_abgabe_4.pdf** im EST ab.

- a) Geben Sie alle Möglichkeiten für die Ausgaben von x und y für einen Aufruf der execute-Methode der Klasse Sync1 an.
- b) Warum sind diese Ausgaben von Sync1 möglich?
- c) Geben Sie alle Möglichkeiten für die Ausgaben von counter für einen Aufruf der execute-Methode der Klasse Sync2 an.
- d) Warum sind diese Ausgaben von Sync2 möglich?



```
public class Sync1 {
       int x = 0; int y = 0;
 3
 4
        class MyThread1 extends Thread{
 5
            public void run() {
 6
                x = 1;
 7
                 System.out.println("y_=_"+y);
 8
Q
        }
10
11
        class MyThread2 extends Thread{
12
            public void run() {
13
                y = 1;
14
                 System.out.println("x_{\perp} = _{\perp}" + x);
15
16
        }
17
        public void execute() throws InterruptedException {
            MyThread1 thread1 = new MyThread1();
MyThread2 thread2 = new MyThread2();
19
20
            thread1.start(); thread2.start();
21
22.
            thread1.join(); thread2.join();
23
24
   }
   public class Sync2 {
2
        Object lock = new Object();
 3
        int counter = 0;
 4
 5
        class MyThread1 extends Thread{
            public void run() {
 6
                 Object myLock = lock;
 7
                 for (int i = 0; i < 2; i++) {</pre>
9
                     synchronized(myLock) {
10
                          counter++;
11
                          myLock = new Object();
12
                     }
13
                 }
14
            }
15
        }
16
        class MyThread2 extends Thread{
17
18
            public void run() {
19
                synchronized(lock) {
20
                     counter++;
21
                 }
22
            }
23
24
25
        public void output() {
26
            System.out.println("counter_=_" + counter);
27
28
29
        public void execute() throws InterruptedException {
           MyThread1 thread1 = new MyThread1();
            MyThread2 thread2 = new MyThread2();
31
32
            thread1.start(); thread2.start();
            thread1.join(); thread2.join();
34
            output();
35
36 }
```