Prof. Dr. Th. Letschert

TECHNISCHE HOCHSCHULE MITTELHESSEN
MNI Mathematik, Naturwissenschaften und Informatik

NVP – Nebenläufige und Verteilte Programme

Aufgabenblatt 4

Aufgabe 1

1. Was ist ein *kritischer Abschnitt*?
2. Was ist eine *atomare Aktion*? Welche Aktionen sind atomar in einem Java–Programm? Ist Atomarität (in Java / auf der JVM) etwas Feststehendes oder ist sie abhängig von der Hardware, dem Betriebssystem, der aktuellen Systemnutzung, ...?
3. Was passiert, wenn zwei Java–Threads gleichzeitig auf die selbe Speicherstelle zugreifen? Ist das überhaupt möglich?
4. Ist es denkbar, dass ein Thread sich zu einem Zeitpunkt in zwei kritischen Bereichen aufhält?
5. Was kooperativ am kooperativem Multitasking?
6. Um was geht es im *Java Memory Model*? Wo findet man genauere Informationen? Ist es auch relevant für Scala-Programme?
7. Wann wird die *IllegalMonitorStateException* geworfen? Konstruieren Sie ein einfaches Beispiel.

Aufgabe 2

1. Gegen Sie jeweils ein Beispiel in Java und Scala an für eine synchronisierte statische Methode und eine synchronisierte nicht-statische Methode.
2. Angenommen ein Thread führt eine synchronisierte nicht–statische Methode einer Klasse `C` aus. Welche Aufrufe kann ein anderer Thread ohne Blockade gleichzeitig ausführen:
   (a) Die gleiche Methode des gleichen Objekts
   (b) Die gleiche Methode eines anderen Objekts der gleichen Klasse
   (c) Andere statische und nicht-synchronisierte Methode des gleichen Objekts
   (d) Andere statische und synchronisierte Methode des gleichen Objekts
   (e) Andere nicht–statische und nicht-synchronisierte Methode eines anderen Objekts der gleichen Klasse
   (f) Andere nicht–statische und synchronisierte Methode eines anderen Objekts der gleichen Klasse

Aufgabe 3

Betrachten Sie folgenden Code:

```
class XX(id: String) extends Thread {
  override def run(): Unit = {
    while (true) m
  }
```

```
  def m: Unit = synchronized {
    println("ENTER thread "+id)
    Thread.sleep(2500)
    println("LEAVE thread "+id);
  }
}

object Aufgabe_3_Main extends App {
  val x1 = new XX("A")
  val x2 = new XX("B")
  x1.start()
  x2.start()
}
```

Wie viele Threads werden bei der Ausführung des Programms mindestens aktiviert? Wird die Methode `m`

- ganz sicher nie,
- eventuell oder
- ganz sicher immer

von mehreren Threads gleichzeitig (nebenläufig) ausgeführt?

Aufgabe 4

Die Variablen `x` und `y` werden in Thread 1 mit Null initialisiert. Dann startet Thread 1 zwei neue Threads: Thread 2 und Thread 3. mit jeweils folgenden Aktionen:

```
Thread 2          Thread 3
x = 1             y = 1
j = y             i = x
```

Kann es passieren, dass i und j den Wert 0 haben? Wenn ja: liegt es daran dass das Verhalten von Java-Programmen nicht eindeutig spezifiziert ist und eine Implementierung – unbeabsichtigt – diversen Unsinn treiben kann, oder liegt es daran, dass ein solches Verhalten explizit von der Java-Spezifikation gedeckt ist und diese einen guten Grund hat, derartiges zuzulassen?

Aufgabe 5

Betrachten Sie die folgenden drei Versionen einer Berechnung:

```
class Version1 {
  val ok = true;

  def work(): Unit = {
     int i = 0;
     while (ok) {
           ....
        i++;
     }
  }

  def stop: Unit = {
     ok = false;
  }

}
```

```
class Version2 {
  @volatile val ok = true;

  def work(): Unit = {
     int i = 0;
     while (ok) {
           ....
        i++;
     }
  }

  def stop: Unit = {
     ok = false;
  }

}
```

```
class Version3 {
  val ok = true;

  def work(): Unit = synchronized {
     int i = 0;
     while (ok) {
           ....
        i++;
     }
  }

  def stop: Unit = synchronized {
     ok = false;
  }

}
```

Die Klasse `Version1` ist problematisch. Warum? Das Problem soll mit `Version2` und `Version3` behoben werden. Welche der beiden Versionen löst das Problem, oder löst es besser als die andere Version?

Aufgabe 6

Welches Verhalten zeigt folgendes Programm:

```
class C {
  val o = new Object();
  def aMethod(): Unit = o.synchronized { wait() }
}

object Aufgabe_6_Main extends App {
  val c = new C();
  c.aMethod();
}
```

Läuft es einfach ohne Probleme zu Ende, verklemmt es (*Deadlock*), kommt es zum Absturz (*Exception*), ...?

Bitte überlegen Sie zuerst und testen dann, ob Ihre Überlegung zutrifft.