Nebenläufige Programmierung

Blatt 1

Name: Bernhard Fritz
Matrikelnummer: 1316136
Gruppe: 11

Gelöste Aufgaben:

1)	23%	/	23%
2)	23%	/	23%
3)	23%	/	23%
4)	30%	/	30%
Gesamt:	100%	/	100%

Starten der Aufgaben:

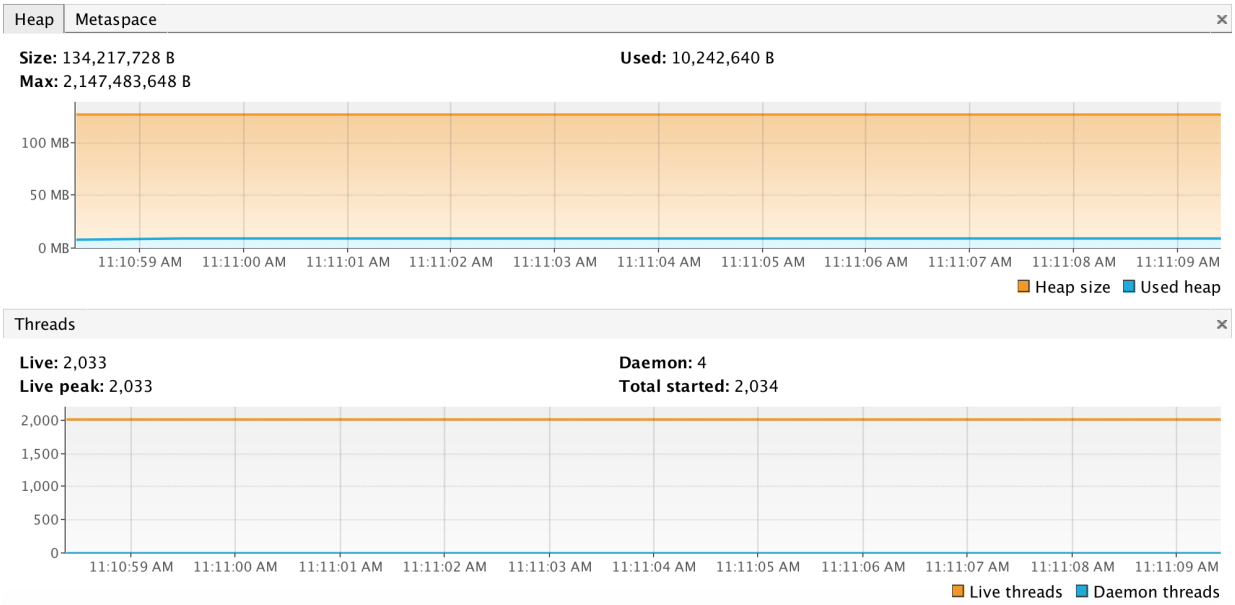
1) ThreadCrash.java
2) TestBench.java
3) ThreadStateMain.java
4) MainTest.java

Aufgabe 1:

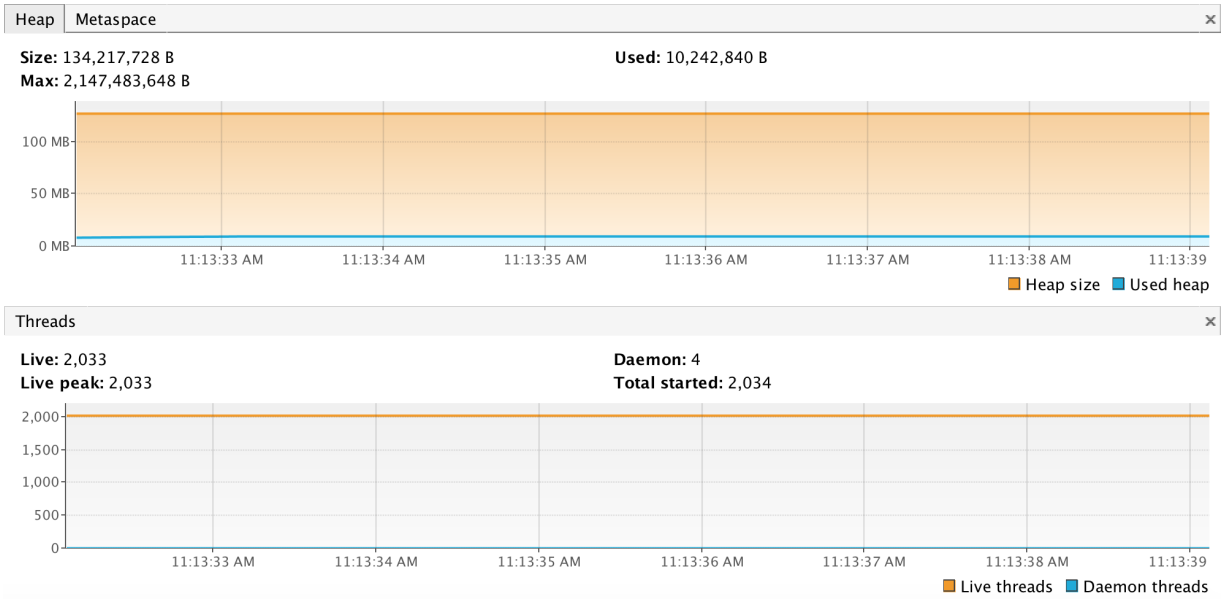
Konfiguration:

- Rechner: Sony MacBook Pro (Retina, 13-inch, Mid 2014)
- Betriebssystem: OS X El Capitan Version 10.11
- Prozessor: 2,8 GHz Intel Core i5
- Speicher: 8 GB 1600 MHz DDR3
- Java Version: 1.8.0_25

Beim ersten Versuch konnten insgesamt **2.034** Threads gleichzeit laufen. Danach wurde das Programm durch eine OutOfMemory Exception gestoppt.



Durch das Java VM – Argument **-Xss160k** wird die Thread-Stacksize auf 160KB gesetzt (das Minimum auf meinem System). Anschließend konnten beim zweiten Versuch **2.034** Threads gleichzeitig laufen, was die selbe Anzahl ist, wie beim Versuch ohne VM – Argument.



Es scheint als ob das Betriebssystem die Anzahl an Threads limitiert. Nach Ausführen des Befehls `sysctl kern.num_taskthreads` folgte folgende Antwort: `kern.num_taskthreads: 2048`
Somit vermute ich, dass sich meine Annahme bestätigt hat und das Betriebssystem ein Limit von maximal 2048 Threads pro Prozess vorsieht.

Aufgabe 2:

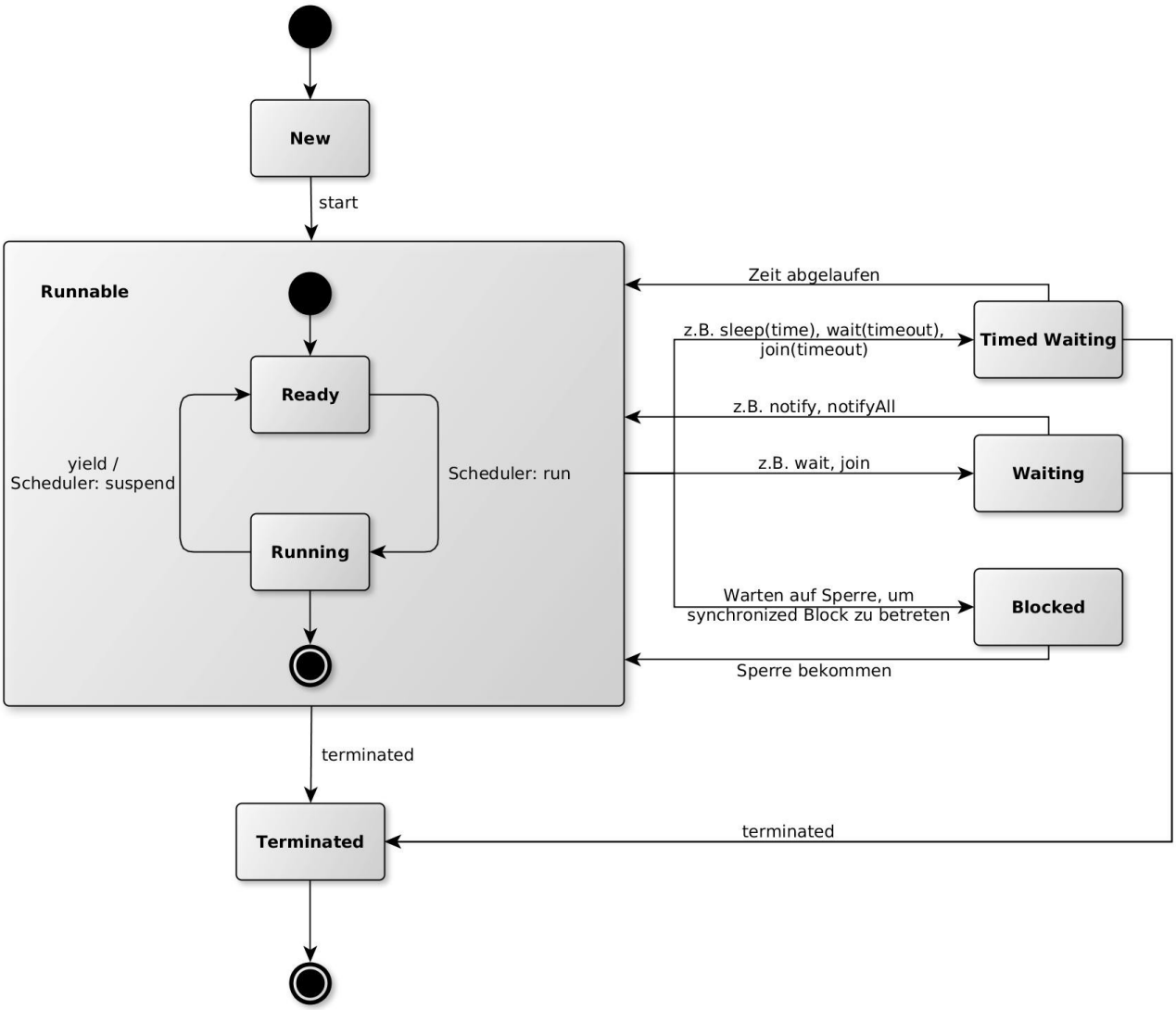
Konfiguration:

- Rechner: Sony MacBook Pro (Retina, 13-inch, Mid 2014)
- Betriebssystem: OS X El Capitan Version 10.11
- Prozessor: 2,8 GHz Intel Core i5
- Speicher: 8 GB 1600 MHz DDR3
- Java Version: 1.8.0_25

Es werden #threads für die Berechnung von n=10.000.000 Iterationen der Formel verwendet.
Die Iterationen werden gleichmäßig auf alle Threads verteilt.
Das Programm wird beendet, wenn die festgelegte Genauigkeit von 7 Kommastellen erreicht ist.

#threads	Zeit[s]
1	40.535
2	20.189
3	18.676
4	17.421
5	17.355
6	17.062
7	17.061
8	17.095
9	17.111
10	17.09
11	16.934
12	16.902
13	17.074
14	16.949
15	16.708
16	16.649
17	16.566
18	16.539
19	16.378
20	16.331

Aufgabe 3:



New: Der Thread wurde erstellt, aber noch nicht gestartet.

Runnable: Auf Betriebssystem-Ebene kann man den JVM-State *Runnable* in die Substates *Ready* und *Running* einteilen. Beim Eintritt in den *Runnable* State befindet sich der Thread im *Ready* State. Bekommt er vom Scheduler CPU-Zeit zugewiesen, wechselt er in den *Running* State. Anschließend bestimmt entweder der Scheduler den Wechsel zurück in den *Ready* State, oder der Thread gibt seine CPU-Zeit mit dem *yield*-Befehl frei.

Timed Waiting: Der Thread wartet eine bestimmte Zeit, bevor er weiter ausgeführt wird.

Waiting: Der Thread wartet auf eine Aktion eines anderen Threads, um fortzufahren.

Blocked: Der Thread wartet auf eine *Sperre*, um einen *synchronized* Block zu betreten.

Terminated: Der Thread hat seine Ausführung beendet.