Nebenläufige Programmierung

Blatt 3

Name:	Bernhard FRITZ
Matrikelnummer:	1316136
Gruppe:	11

Gelöste Aufgaben

1)	45%	/	45%
2)	30%	/	30%
3)	25%	/	25%
=	100%	/	100%

Starten der Aufgaben

1) TestMainC, TestMainE
2) MainRaceConditions, MainThreadSave
3) Benchmark

Aufgabe 1

a)

Eine Listener Registrierung direkt im Konstruktor benötigt die Verwendung der this Referenz im Konstruktor. Dies sollte man vermeiden, weil man eine Referenz auf ein noch nicht fertiges Objekt freigibt. Damit hat ein anderer Thread die Möglichkeit, das noch nicht fertige Objekt zu verwenden. Dies kann zu Problemen wie beispielsweise den Zugriff auf noch nicht allokierten Speicher führen.

f)

Immutable Klassen haben den Vorteil, dass deren Attribute nach Erstellung des Objektes nicht mehr verändert werden können. Dies kann dann sinnvoll sein, wenn man sicherstellen will, dass weitergegebene Objekte von anderen nicht mehr manipuliert werden können. Die Objekte werden dann nur zum Lesen freigegeben.

Aufgabe 2

b & c)

Das Problem der *NumberRange* – Implementierung ist, dass die Check für die Invariante und das Setzen des neuen Lower/Upper – Wertes nacheinander stattfindet und nicht Thread save ist.
So kann es beispielsweise zu folgender Ausgabe kommen:

```
UpperThread1: Set upper to 61
LowerThread1: Set lower to 89
InRangeThread1: 85 is not in range (89 – 61)
```

Wie man sehen kann, ist die Invariante (lower <= upper) verletzt (89 > 61).

Durch die Änderung, dass der *LowerThread* bzw. *UpperThread* nach dem Check kurz wartet (1ms), und damit dem jeweils anderen Thread die Chance zur Ausführung gibt, könnten beide Threads Operationen auf Variablen ausführen, die nicht mehr die Werte besitzen, die sie bei der Überprüfung der Invariante hatten.
Generell sollten Checks und Aktionen auf Attributen nie in einem ungesicherten Block passieren, weil zwischen diesen beiden Statements andere Threads zur Ausführung kommen könnten, die ebenfalls diese Attribute verwenden / verändern.

d)

Um NumberRange Thread save zu machen, kann man deren Methoden um das synchronized - Attribut erweitern. Damit ist sichergestellt, dass zwischen Check und dem Schreiben auf die Attribute kein anderer Thread Änderungen daran vornehmen kann.

Aufgabe 3

c)

Aus den Ergebnissen (siehe unten) ist ersichtlich, dass das Einfügen von Daten in eine HashMap deutlich aufwändiger ist, als das Lesen. Der Grund dafür ist vermutlich, dass beim Einfügen eine HashCollision auftreten kann, die dann dementsprechend erst behandelt werden muss.

MacBook Pro (Retina, 13-inch, Mid 2014)
Processor: 2,8 GHz Intel Core i5
Memory: 8 GB 1600 MHz DDR3
OS: OS X El Capitan Version 10.11.1
Java: Version 1.8.0_25

Without synchronization:
==========
# of threads | HashMap accesses per second (50% insert, 25% read, 25% remove)
---
1 | 2127660
2 | 3448276
4 | 3225806
8 | 1975309
16 | 1886792

With synchronization:
==========
# of threads | HashMap accesses per second (50% insert, 25% read, 25% remove)
---
1 | 3448276
2 | 5263158
4 | 3636364
8 | 2564103
16 | 2292264

Concurrent:
==========
# of threads | HashMap accesses per second (50% insert, 25% read, 25% remove)
---
1 | 3846154
2 | 3508772
4 | 2531646
8 | 1985112
16 | 1963190

Without synchronization:
==========
# of threads | HashMap accesses per second (80% insert, 15% read, 5% remove)
---
1 | 1428571
2 | 2469136
4 | 1342282
8 | 1152738
16 | 1075992

With synchronization:
==========
# of threads | HashMap accesses per second (80% insert, 15% read, 5% remove)
---
1 | 3125000
2 | 2298851
4 | 1587302
8 | 1307190
16 | 1211204

Concurrent:
==========
# of threads | HashMap accesses per second (80% insert, 15% read, 5% remove)
---
1 | 3125000
2 | 2898551
4 | 1731602
8 | 1372213
16 | 1222307

Without synchronization:
==========
# of threads | HashMap accesses per second (5% insert, 90% read, 5% remove)
---
1 | 2702703
2 | 4878049
4 | 1101928
8 | 911162
16 | 973828

With synchronization:
==========
# of threads | HashMap accesses per second (5% insert, 90% read, 5% remove)
---
1 | 3846154
2 | 2666667
4 | 1139601
8 | 988875
16 | 1021059

Concurrent:
==========
# of threads | HashMap accesses per second (5% insert, 90% read, 5% remove)
---
1 | 4166667
2 | 2531646
4 | 1183432
8 | 996264
16 | 1003764