Übung 5: Thread Pools

Zielsetzung:

- Thread Pools zur algorithmischen Parallelisierung anwenden.
- Asynchrone Programmierung mittels Thread Pools realisieren.

Aufgabe 1: Fork Join Thread Pool

Quicksort ist nicht nur im Durchschnittsfall einer der schnellsten Sortieralgorithmen, sondern er lässt sich auch optimal parallelisieren. Der sequentielle Quicksort ist in der Vorlage implementiert:

```
// sorts the entire integer array in ascending order
void sort(int[] array) {
  quickSort(array, 0, array.length - 1);
}

// sorts the partition between array[left] and array[right]
void quickSort(int[] array, int left, int right) {
  // split into two partitions
  int i = left, j = right;
  int m = array[(left + right) / 2];
  do {
    while (array[i] < m) { i++; }
    while (array[j] > m) { j--; }
    if (i <= j) {
      int t = array[i]; array[i] = array[j]; array[j] = t;
      i++; j--;
    }
  } while (i <= j);
  // recursively sort the two partitions
  if (j > left) { quickSort(array, left, j); }
  if (i < right) { quickSort(array, i, right); }
}
```

a)

Realisieren Sie einen parallelen Quicksort-Algorithmus unter Verwendung des Java Fork-Join Thread-Pools. Die Teilsortierungen der linken und rechten Partition sollen als neue rekursive Tasks ausgeführt werden, sofern die beiden Teilpartitionen genügend gross sind. Sie können mit dem sequentiellen Quicksort aus der Vorlage beginnen.

Hinweis: Mit `invokeAll` können Sie direkt eine Anzahl Tasks starten und deren Beendigung abwarten:

```
a.fork(); b.fork(); b.join(); a.join();
```

ist (konzeptionell) äquivalent zu:

```
invokeAll(a, b);
```

b)

Vergleichen Sie die Performance zwischen dem parallelen und dem sequentiellen Quicksort. Tunen Sie den Schwellwert für Sub-Tasks (Grösse der Teil-Partitionen) so, dass es einen maximalen Speedup erreicht. Was für einen Speedup erreichen Sie? Mit welchem Schwellwert ist der parallele Quicksort schneller bzw. langsamer als die sequentielle Variante?

Aufgabe 2: Asynchrone Downloads

In der Vorlage finden Sie ein einfaches Testprogramm für den Download einer Menge von Webseiten. Ein solcher Download soll aber besser nicht-blockierend sein, d.h. alle Downloads können gleichzeitig laufen.

Implementieren Sie zuerst eine asynchrone Download-Methode mit folgender Signatur:

```
CompletableFuture<String> asyncDownloadUrl(String link)
```

Bauen Sie das Testprogramm so um, dass alle Downloads gleichzeitig laufen und die Gesamt-Downloadzeit verkürzt wird.

Realisieren Sie dies mit zwei verschiedenen Ansätzen:

a) Caller-zentrisch

Die main-Methode wartet auf die Beendigung der zurückgegebenen Futures der asynchronen Download-Methode.

Verwenden Sie hierfür `get()` der Klasse `CompletableFuture`.

b) Callee-zentrisch

Jeder asynchrone Download löst bei seiner Beendigung selbständig eine Continuation aus, welche die Downloadzeit ausgibt. Zudem soll die Gesamtzeit am Schluss ausgegeben werden.

Verwenden Sie hierfür `thenAccept()` und `allOf()` der Klasse `CompletableFuture`.