OCP Java SE 8

Streams

Parallel Stream

- parallel() auf einem Stream
- parallelStream() auf einer Collection
- isParallel() auf einem Stream

Parallel Stream

- forEach () auf einem parallelen Stream ist gleichbedeutend mit dem Senden mehrerer Runnable Lambda Ausdrücke zu einem Thread Executor Pool.

Parallel Stream

- forEachOrdered()
- Reihenfolge wird auf Kosten von Performance beibehalten

Parallel Streams Performance

- Deutlich bessere Performance bei großen Datenmengen
- Bei kleinen Datenmengen sind sequenzielle Streams performanter

Parallel Stream

- Parallel streams können die Performance signifikant verbessern, wenn die auszuführenden Operationen independent sind.

Parallel Stream

- Independent Operations
  - arbeiten isoliert auf einem Element und benötigen keine Informationen über andere Elemente
- Stateful Operations
  - erfordern die Kenntnis der anderen Elemente im Stream (oder zumindest eines Teils davon)

Parallel Stream

- Arbeitet ein Parallel Stream mit einer Collection ist es wichtig, dass diese Threadsave ist.
- Kann sonst zu Race Conditions kommen, so dass Daten verloren gehen

Parallel Reductions

- Ergebnisse können zwischen seriellen und parallelen Streams abweichen

Order-Based Tasks

- Ergebnisse können zwischen seriellen und parallelen Streams abweichen
- Order–Based Operationen arbeiten langsamer

Unordered Streams

- Durch unorderd() kann die Reihenfolge bei Order–Based Operationen ignorieren

reduce (Immutable Reduction)

- identity
- accumulator
- combiner
- z.B:
  stream.reduce("", String::concat, String::concat);

collect (Mutable Reduction)

- supplier
- accumulator
- combiner
- z.B:
  stream.collect(ArrayList::new, List::add,
  List::addAll);

collect

- flatMap () erstellt einen neuen Stream, der standardmäßig nicht parallel ist, unabhängig davon, ob die zugrunde liegenden Elemente parallel waren
- concat() produziert einen parallelen Stream wenn zuvor eines der Teile parallel war