Parallel Programming in Java

A Tour of the Java Concurrency API

Christian Heitzmann | SimplexaCode AG

SimplexaCode

SimplexaCode AG | Grimselweg 11/501 | 6005 Lucerne | Switzerland
+41 43 810 06 03 | *info@simplexacode.ch* | *www.simplexacode.ch*

Java Vienna Meetup | TU Wien | Austria
March 11, 2024

Why Concurrency Matters to Every (Java) Programmer

... and is more than just performance:

- multicore processors
- asynchronous events
- simplification
- Java frameworks
- graphical user interfaces (GUIs)
- Java virtual machine (JVM) background threads

$\Longrightarrow$ Thread safety is crucial.

Shortlinks

- these slides and source code examples: ***link.simplexacode.ch/9y78***
- latest version of *Thread Visualizer*: *link.simplexacode.ch/cpiw*

Demo | First Steps in *Thread Visualizer*

1. *[...]*.a_introduction.**A_FirstSteps**
2. *[...]*.a_introduction.**B_SingleThread**

Demo | Interrupting Threads

1. *[...]*.b_interrupting_threads.**A_InterruptibleThreadClasses**
2. *[...]*.b_interrupting_threads.**B_InterruptibleExecutor**

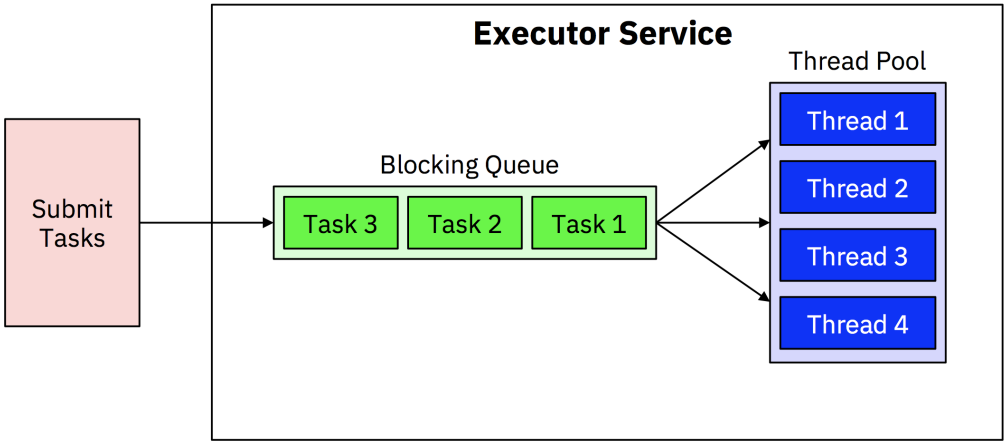
Interrupting Threads

- ordinary shutdown
  - ≥ Java 5: `shutdown()`
  - ≥ Java 19: `close()` (via `AutoClosable`)
  - non-blocking
  - Finishes all submitted tasks, but does not accept new tasks.
- immediate shutdown
  - `shutdownNow()`
  - non-blocking
  - Interrupts all executing tasks, but does not guarantee their termination.
- awaiting termination
  - `awaitTermination(long, TimeUnit)`
  - blocking
  - `throws InterruptedException`

Take-Home Messages | Interrupting Threads

1. Prefer `Executors` over `Threads`.
2. Consider interrupting child threads if interrupted.
3. Shut down `ExecutorServices` after use and after interrupt.

Executor Services and Thread Pools



Demo | Executor Types

[...].c_executors_and_pools.**A_ExecutorTypes**

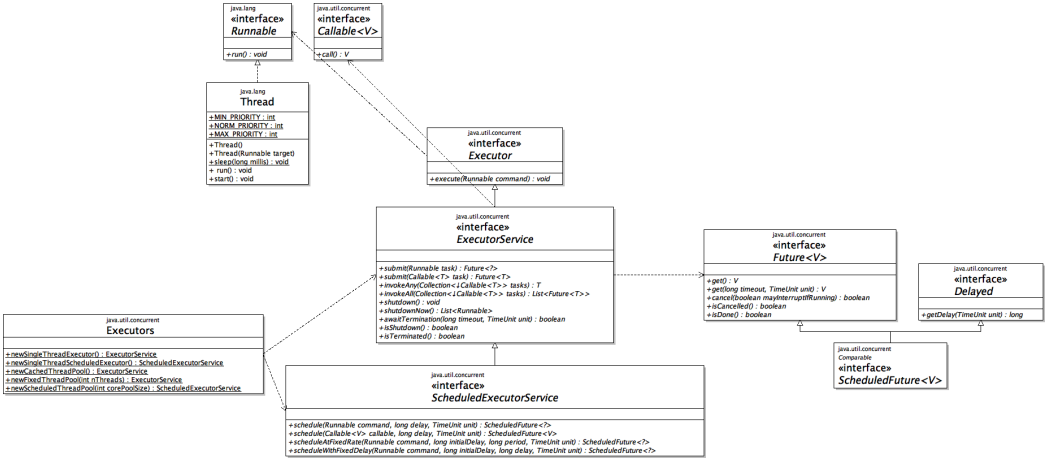
1. `newSingleThreadExecutor()`
2. `newFixedThreadPool(2)`
3. `newFixedThreadPool(`*# CPUs*`)`
4. `newCachedThreadPool()`
5. ≥ Java 21: `newVirtualThreadPerTaskExecutor()`

Demo | Scheduled Executor

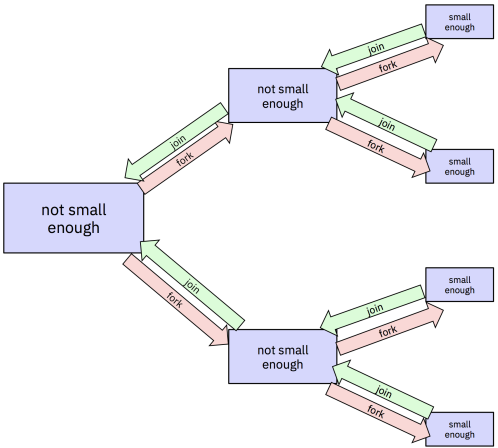
[...].c_executors_and_pools.**B_ScheduledExecutor**

1. `schedule(Runnable/Callable, long, TimeUnit)`
2. `scheduleWithFixedDelay(Runnable, long, long, TimeUnit)`
3. `scheduleAtFixedRate(Runnable, long, long, TimeUnit)`

Executor Framework



Fork/Join Framework



Covered in great detail in

- English blog post
  "The Fork/Join Framework"
  - *link.simplexacode.ch/529z*
- German JavaSPEKTRUM article
  "Teile und herrsche – Das Fork-Join-Framework"
  - *link.simplexacode.ch/vidj*

Demo | Fork/Join Framework

[...].c_executors_and_pools.**C_ForkJoinPool**
[...].c_executors_and_pools.**C_ForkJoinPoolAction**

Take-Home Messages | Executors and Pools

1. Study, evaluate, and practice all relevant interfaces and classes.
2. Have a detailed plan or leave it up to experts.
3. Keep it simple and consistent.

Incrementing Counter

What is the final result of `counter`?

```
public final class IncrementingCounter
    extends AbstractComputation {

  private long counter = 0;

  // separators omitted

  private void incrementCounterInLoop() {
    for (long i = 1; i <= 100_000_000; i++) {

      // interrupted check omitted

      counter++;
    }
  }
```

```
@Override
public void run() {
  ExecutorService service
      = Executors.newFixedThreadPool(8);

  for (int threadNumber = 1;
       threadNumber <= 8;
       threadNumber++) {

    service.execute
        (() -> incrementCounterInLoop());
  }

  // service shutdown omitted

  System.out.format("counter = %,d%n", counter);
}
```

Demo | Incrementing Counter

1. *[...]*.d_locks_and_concurrent_classes.
   A_IncrementingCounter
2. *[...]*.d_locks_and_concurrent_classes.
   B_IncrementingAtomicCounter

Compound Actions

- read-modify-write:

```
counter++;
```

$\Longrightarrow$

```
long oldValue = counter;
long newValue = oldValue + 1;
counter = newValue;
```

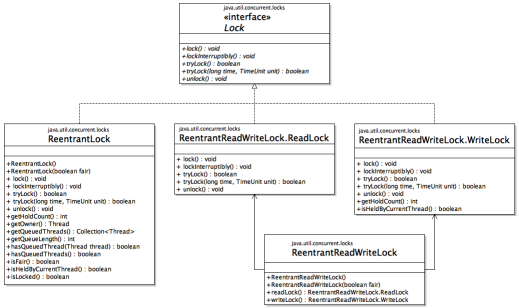
- check-then-act:

```
if (instance == null) {
  instance = new MyObject();
}
return instance;
```

- put-if-absent
- remove-if-equal
- replace-if-equal
- iteration
- navigation

Locks

- `synchronized`
  (= `synchronized (this)`)
- `synchronized (Object)`
- `ReentrantLock`
- `ReentrantReadWriteLock`

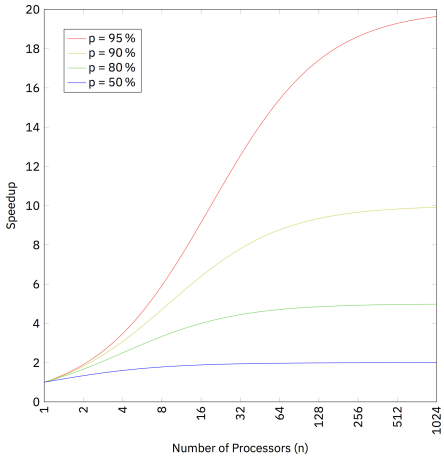


Locking, Serialization, and Contention

- Locks and `synchronized` blocks are not expensive by themselves; the resulting contention is.
- Amdahl's Law:

$$speedup = \frac{1}{(1-p) + \frac{p}{n}}$$

p: parallel portion
n: number of processors



Demo | Concurrent Collections

[...].d_locks_and_concurrent_classes.**C_ConcurrentCollections**

Concurrent Collections

- Queues
  - ⟹ ArrayBlockingQueue
  - ⟹ ConcurrentLinkedDeque
  - ⟹ ConcurrentLinkedQueue
  - ⟹ DelayQueue
  - ⟹ LinkedBlockingDeque
  - ⟹ LinkedBlockingQueue
  - ⟹ LinkedTransferQueue
  - ⟹ PriorityBlockingQueue
  - ⟹ SynchronousQueue
- Lists
  - ArrayList
    ⟹ CopyOnWriteArrayList
- Sets
  - HashSet
    ⟹ CopyOnWriteArraySet
- SortedSets
  - TreeSet
    ⟹ ConcurrentSkipListSet
- Maps
  - HashMap
    ⟹ ConcurrentHashMap
- SortedMaps
  - TreeMap
    ⟹ ConcurrentSkipListMap

Take-Home Messages | Locks and Concurrent Classes

1. Watch out for (hidden) compound actions.
   - Either replace them by atomic data structures or methods
   - or guard them by locks.
2. Keep the scope of synchronized blocks small, but not too small.
3. Prefer concurrent classes over synchronized wrapper classes.

Take-Home Messages | Conclusion

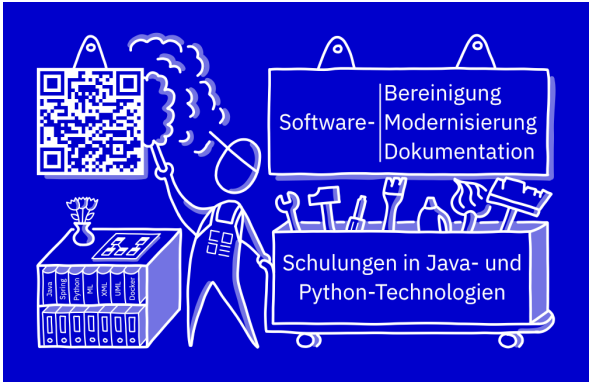
1. If you ever find yourself tweaking thread priorities, garbage collectors, or delay routines: Stop!
2. Understand and use the higher-level utility classes of the Java Concurrency API.
3. Keep it simple.

Articles and Training

- JavaSPEKTRUM articles by me about parallel programming
  - *"Aus eins mach zwei – Was ist eigentlich ein Spliterator?"*
    - *link.simplexacode.ch/srxg*
  - *"Nebenläufige Programmierung mit Zukunft? – Future und CompletableFuture"*
    - *link.simplexacode.ch/hsq0*
  - *"Du kommst hier nicht rein! – Locking in Java"*
    - *link.simplexacode.ch/srgm*
  - *"Teile und herrsche – Das Fork-Join-Framework"*
    - *link.simplexacode.ch/vidj*
- SimplexaCode trainings
  - Parallel Programming in Java, Machine Learning with Python, Switching to Python, Documentation, Docker, Spring (Boot)
    - *www.simplexacode.ch → Schulungen in Java und Python-Technologien*

Questions?

SimplexaCode



link.simplexacode.ch/9y78