# Java Concurrency

If it’s worth If it’s doing, worth doing, it’s worth it’s worth doing doing right. right.

Mike Christianson http://codeaweso.me

# Primitives

- `Thread`
- `synchronized`
- `wait/notify`
- Collections

# Synchronized

- **statement**: lock on object given in statement
- **method**: lock on object instance
- **static method**: lock on class

# Primitives

- low-level
- prone to correctness and performance issues
- encourage violation of DRY
- encourage violation of SRP

# Collections

- Synchronized Collections
  - thread-safe operations
  - *iteration* requires explicit/external synchronization
  - `ConcurrentModificationException`
  - poor performance across multiple threads
- Unmodifiable Collections

# Newer concurrency tools

- Executors
- Synchronizers
- Concurrent Collections
- Locks
- Atomic Variables

# Executors (Java 5)

- Variations
  - single-thread
  - fixed-size threadpool
  - scheduled
- Execution
  - always returns a `Future`
  - `Future` may contain a result

# Synchronizers (Java 5)

- Semaphore
- CyclicBarrier
- CountDownLatch
- Exchanger

# Collections (Java 5)

- BlockingQueue
- ConcurrentMap
- CopyOnWriteArrayList
- PriorityBlockingQueue
- BlockingDeque (Java 6)

# Locks (Java 5)

- ReentrantLock
- ReentrantReadWriteLock

# Atomic Variables (Java 5)

- AtomicBoolean
- AtomicInteger
- AtomicReference
- AtomicStampedReference

# Newest concurrency tools

- **Executors**: Fork/Join (Java 7)
- **Collections**: TransferQueue (Java 7)
- **Message Passing**: Actors (e.g., *Akka*; or *Scala*)
- **Transactional**: STM (e.g., *Multiverse*; or *Clojure*)
- **Functional Reactive**: e.g., RxJava

# Future concurrency tools (Java 8)

- Lambda Expressions (closures)
- Parallel Collections
- LongAdder/DoubleAdder