KU LEUVEN

Programmeertechnieken/Programming Techniques

Part 6
Concurrency

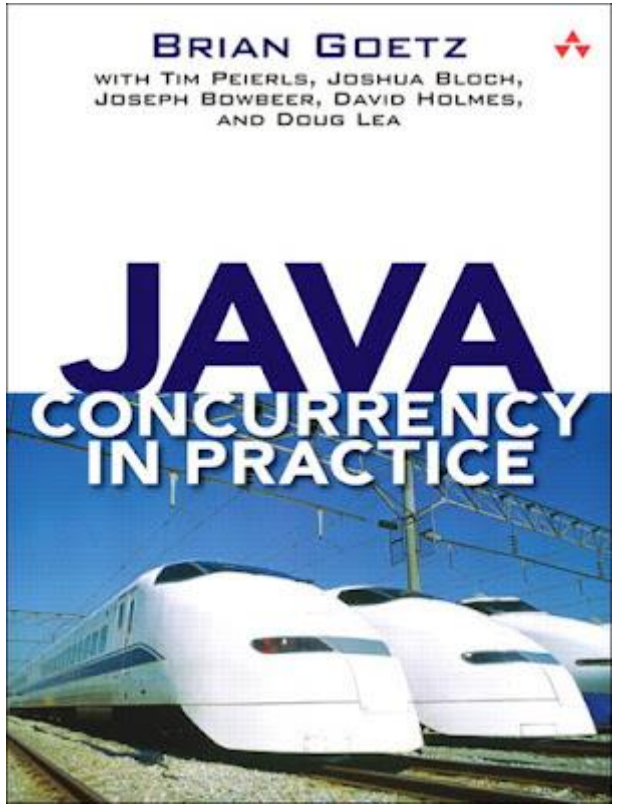
Koen Pelsmaekers

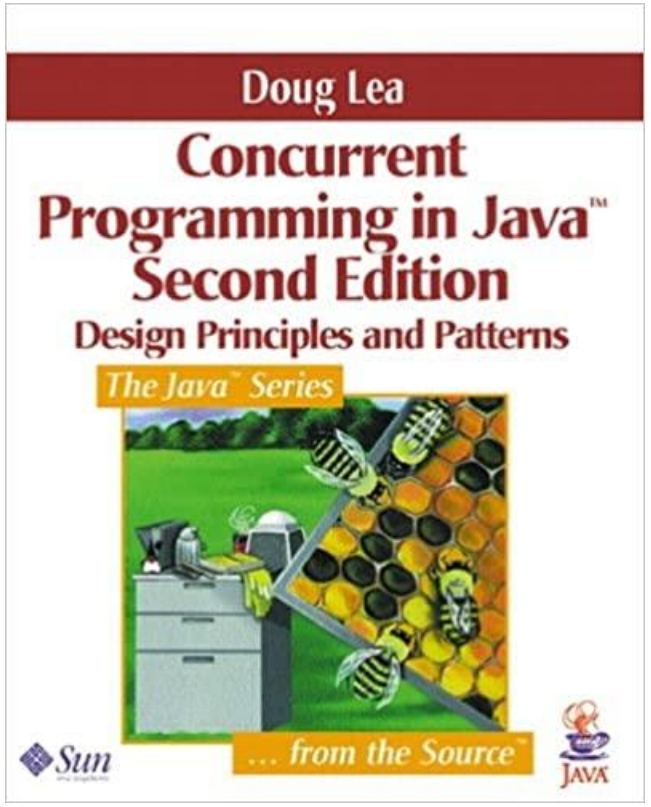
Campus Groep T, 2022-2023

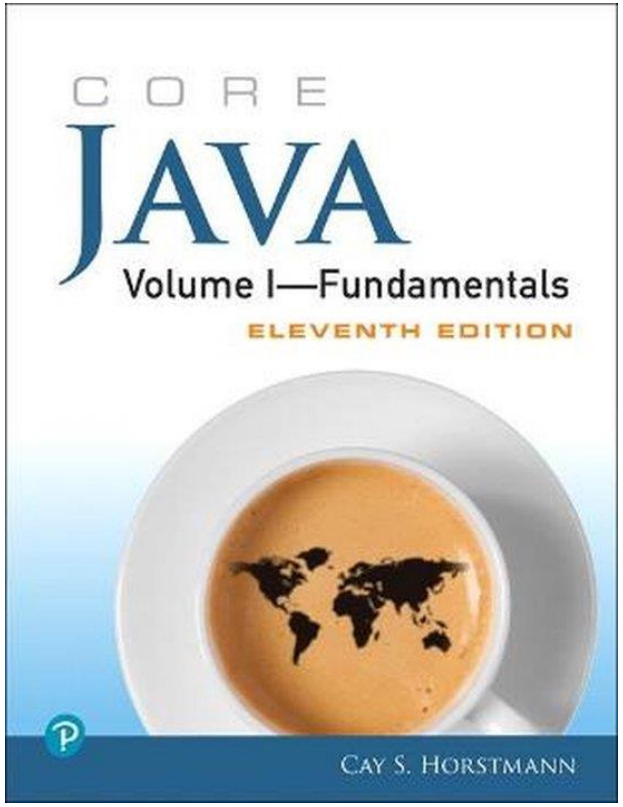
Intro to concurrency

- Concurrent software
  - Computer can run multiple tasks at the same time: **processes & multitasking**
  - A single application can do more than one thing at the same time: **threads & multithreading**
- Java supports writing concurrent software at different levels of abstraction

Sources for examples…



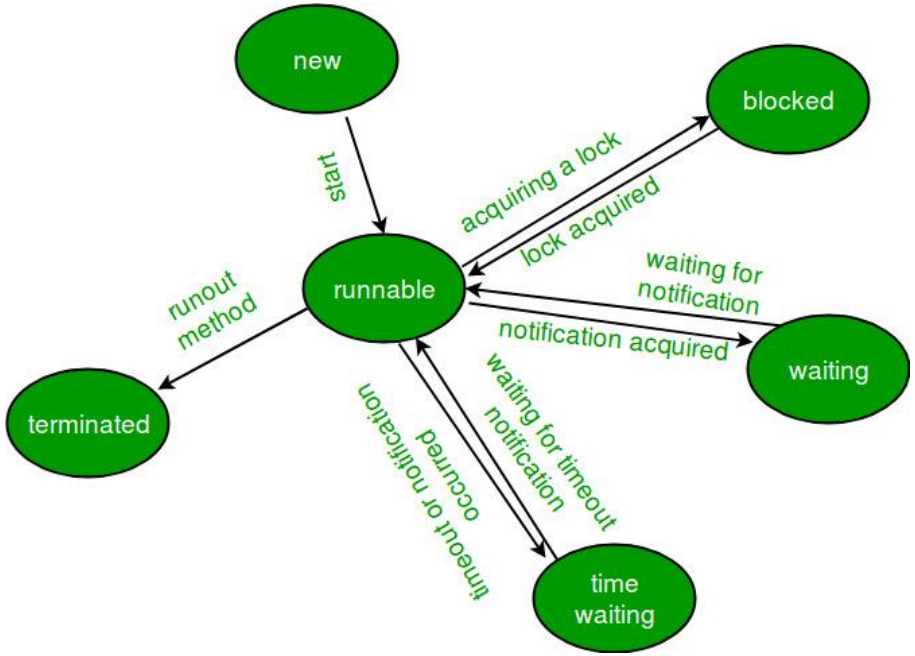




Introduction to multithreading

- Available processors?
- Multithreading example
  - ball animation example(s): every ball runs in its own thread
- Simple Thread example
  - subclass Thread and override run() (example MessageThread1) or provide a Runnable implementation as parameter to Thread constructor (example MessageThread2 +lambda)
  - call “start()” method to start a new thread (do not call the “run()” method to start a thread!)

Thread states



A thread state. A thread can be in one of the following states:

- `NEW`
  A thread that has not yet started is in this state.
- `RUNNABLE`
  A thread executing in the Java virtual machine is in this state.
- `BLOCKED`
  A thread that is blocked waiting for a monitor lock is in this state.
- `WAITING`
  A thread that is waiting indefinitely for another thread to perform a particular action is in this state.
- `TIMED_WAITING`
  A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.
- `TERMINATED`
  A thread that has exited is in this state.

Threads in the runnable state can be selected by the thread scheduler (priorities?) => “Running”

Thread properties

do not use run() but start → set main thread and after finished, others start

- name and id
- priority (maps operating system thread priorities)
  - MIN_PRIORITY, MAX_PRIORITY, NORM_PRIORITY
  - look out for “starvation” (thread with low priority never executes)
- daemon threads
  - t.setDaemon(true);
  - for example timer threads
  - when only daemon threads are running => VM exits
- ThreadGroup

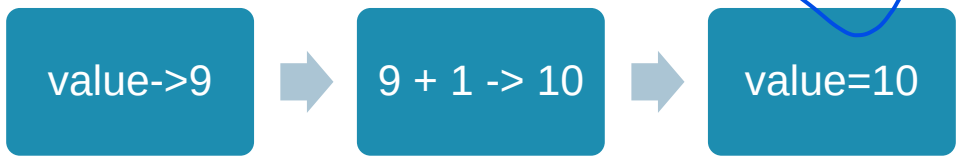
Concurrency issues

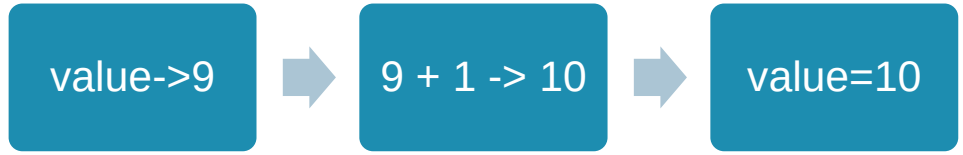
- Thread-safe code?

  managing *"shared, mutable state"*
  - shared = variable could be accessed by multiple threads
  - mutable = value of variable could change during its lifetime

- "Race condition" example: UnsafeSequence
  - problems: missing numbers? duplicate numbers?
  - compound actions = "read-modify-write" or "check-then-act" problem

  => atomic operation?





Race condition: solutions

AtomicInteger

- Race condition: “when the correctness of a computation depends on the relative timing or interleaving of multiple threads by the runtime”
  (sometimes or most of the times it will work?)
- “Critical section(s)” in code
- Solutions
  - java.util.concurrent.atomic package: AtomicSequence
    - solutions for a single “element of state”
  - locking (“synchronize” read/write access with a synchronized block)
    - lock hold by an object, works like mutexes: at most one thread may own a lock, other thread(s) acquiring the lock will have to wait (“blocked”)
    - to “serialize” access to critical sections
    - be aware: every access (read and write) to the same value, should be guarded by the same lock!

Synchronizing threads

- Synchronize with “synchronized” on intrinsic lock object
- Synchronize with Lock/Condition
- java.util.concurrent package

  blocking queues, concurrent collections, timers, …
- java.util.concurrent.atomic package

  incrementAndGet(), compareAndSet(), updateAndGet(), accumlateAndGet(), …

Callables & Futures

- Runnable = task that runs asynchronously
  - No parameters
  - No return value
- Callable = Runnable + returns a value (CallableFutureDemo)
  - Interface with one method: `V call() throws Exception;`
- Future = result of an asynchronous computation
  - `get()` waits until result is available
    - get with a timeout
  - `isDone()`
  - `cancel()`

Executors Framework

- Seperates thread creation and management
  - Thread creation: pool of threads
  - Thread management: lifecycle management of threads
  - Task submission and execution
- Executor, ExecutorService, ScheduledExecutorService interfaces + their implementations (ExecutorsExample and ExecutorsExample2)
  - SingleThreadExecutor, FixedThreadPool, …
  - static factory methods in Executors class
- Example: count number of occurences in files (from Core Java)
  ExecutorReadFileExample/Matchcounter

Fork/Join

- Used under the hood by parallelstream
- uses “Work stealing”

Lambda expressions

① take some time
② a large amount data

- stream() vs. parallelStream()
- see ParallelStreamDemo