Threads and concurrency: what problems arise?

● Concurrent access to shared resources can cause problems, the computer executes very quickly, sequential intuition goes out the window
  ➤ see the Producer/Consumer example for what can happen
  ➤ complete understanding includes starvation, fairness, deadlock
    • starvation: does some thread never execute
    • fairness: do all threads eventually get a chance to execute
    • deadlock: are all threads running, but no progress made

● Operating Systems and Dining Philosophers
  ➤ five people in a circle, chopstick between each person, algorithm to ensure that everyone thinks and eats (pick up one chopstick at a time)
Java threads, Platform problems

- In general avoid tight thread loops that only compute
  - use `Thread.sleep()`, use `Thread.yield()`
    - yield explicitly gives other threads a chance to execute, but doesn’t pause

- On Solaris machines there’s no guarantee (not required in Java) that threads scheduled with time-slicing, so use yield to ensure some degree of fairness

- Try to avoid dealing with shared resources as in Producer/Consumer problem, hard to get right
  - Scooter example: what do robots do?
    - Queue up at destination: factory/robot
    - Is the destination threaded? Who processes arrivals and departures?
Monitors, condition variables

- All objects with at least one synchronized method share a monitor (concept due to C.A.R. Hoare)
  - access to synchronized methods requires acquiring a lock via the monitor, only one synchronized method is executed at a time on a *per-object* basis
  - no process/Thread should hold the lock indefinitely, it’s possible to wait() when the lock is held, relinquishing monitor to other waiting threads, then re-aquire the lock
  - when a synchronized method finishes, the lock is lost, except for recursive calls --- monitors are *re-entrant*
  - see Nutshell and Core v. II, in general when using wait(), use notifyAll() rather than notify()
  - interfaces can’t be synchronized, see Producer/Consumer
Scooter Threads

- If each robot is a thread, what does the thread do?
  - thread could move robot
  - thread could draw robot (or both)
  - advice: let one thread deal with all drawing, ok to have robots update moves in separate threads

- What about a factory? Is it multithreaded?
  - Two tasks: create boxes, unload boxes. Is this a producer/consumer problem? Are there other possibilities?

- What about a rocket? Is this a problem?
  - What does a rocket do?

- Can we fix all these problems by having the robots all move/paint in the same thread?