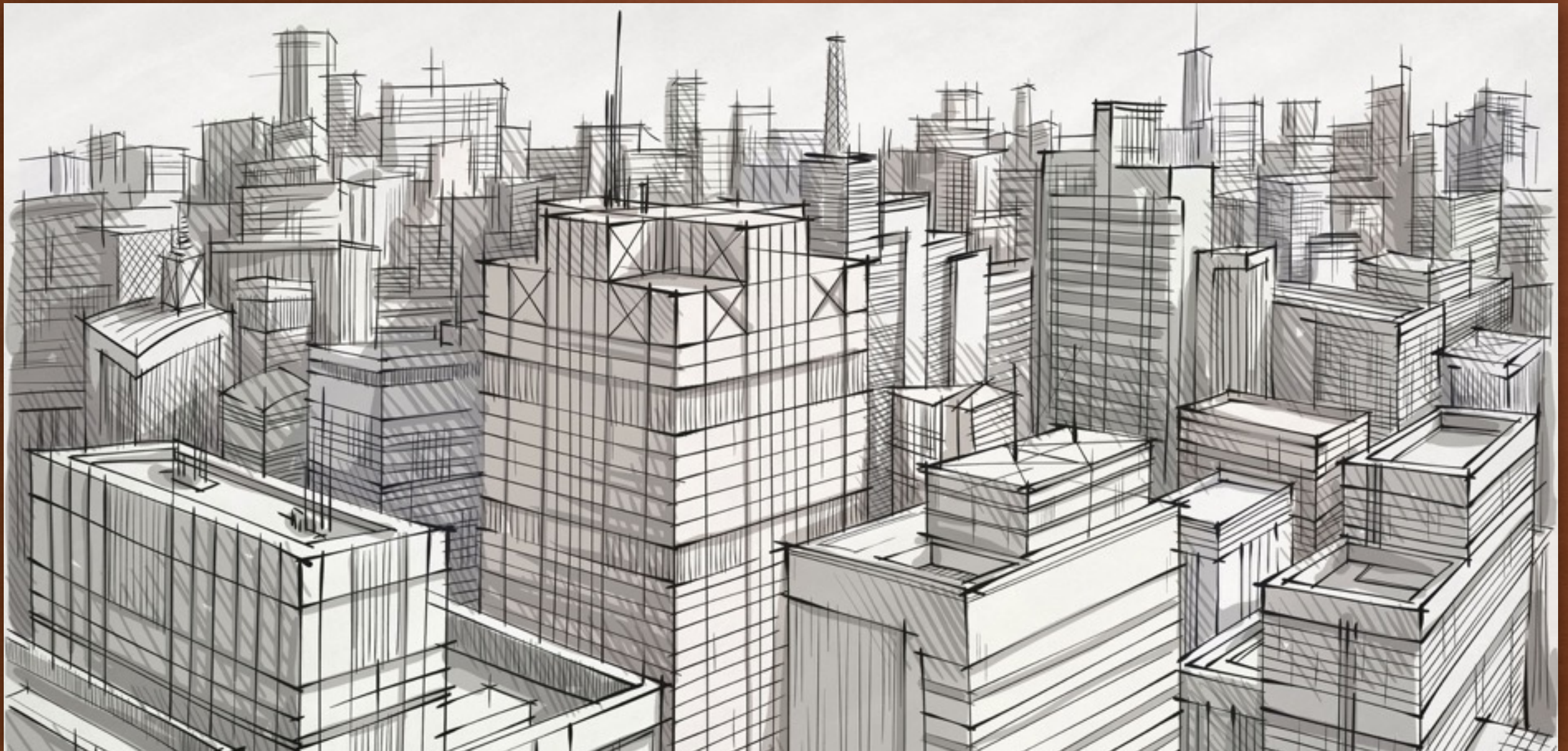


CS1530, LECTURE 16:
CONCURRENT PROGRAMMING IN JAVA
BILL LABOON



LAST CLASS

- We discussed -
 - The benefits and drawbacks of multi-threaded programming
 - Thread usage in Java
 - Thread and Runnable classes, synchronization
 - Issues with threading (data races, deadlock, livelock)
 - Some attempts to ameliorate them (lock hierarchy, analysis, randomized polling)

THIS CLASS

- Using the Java library to avoid many of those issues
- Specifically, many helpful classes in `java.util.concurrent`

ATOMIC VARIABLES

- Variables that can be accessed atomically
 - That is, can do complex operations such as increment atomically
 - Avoids data races
- Built-in synchronization
- Atomic.java

COLLECTIONS

- Some built-in collections are thread-safe, some are not
- HashMap vs Hashtable
 - HashMap IS NOT thread-safe
 - Hashtable IS thread-safe
 - But really you should start using
- Be sure to look at API description to determine which you are using!

THREAD SAFETY IS NOT FREE

- Usually performance decreases, even in “non-threaded” applications
- Remember that if you are programming in Java, you are actually ALWAYS dealing with a threaded application running on the JVM
- Try timing HashMap.java vs Hashtable.java

BUT DOES IT MATTER?

- If you are not sharing mutable state, no
- The garbage collector is not going to come and mess up your variables
- Otherwise 401 would have been MUCH more difficult

LET THE JAVA.UTIL.CONCURRENT DESIGNERS WORRY

- “Don’t roll your own crypto”
- Don’t do more synchronization than you have to
- Most common data structures already have thread-safe version
- `java.util.concurrent` package in Java

JAVA.UTIL.CONCURRENT EXAMPLE

- Semaphore
- ThreadPoolExecutor
- Blocking Queue
- CountdownLatch
- Copy-on-Write ArrayList