What did we talk about last class?
Motivation

- The future is parallel.
- Core counts are growing but clock speed isn't and neither is single thread performance.
- Software developers are behind the curve on this.
Basic Approach

- You can use the `java.lang.Thread` class to represent a thread.
- Pass it a new `Runnable` that you define a `run` method in and call `start` to make it go.
- This makes it very easy to start new threads, but there are significant pitfalls when mutable memory is involved.
The join method of Thread will block until that thread has finished working.

This is something you can do when you want a computation to continue only after each of the threads has completed.

This only works if you are completely done with those threads.
Synchronization

- Threads use shared memory and you don't get significant control over what happens when.
- Race conditions are errors that occur because of dependence on timing details.
- Bank example.
- You can synchronize on objects to make sure critical blocks aren't accessed in parallel
  - `obj.synchronized { ... }`
- Slow and can cause deadlock.
wait/notifyAll

- Allows synchronization between threads. A thread can wait and it won't restart until another thread notifies it.
- Put wait in while loop that checks boolean.
- Always use notifyAll instead of notify. Failure to do so leads to deadlocks.
I want to get more working including commands working so that we can play with some of this in the drawing program.
How many cores does your computer have? Have you ever tried to keep them all busy?

The next IcP is Wednesday.