Application Design Using Java

Lecture 13

Callables and Futures

- Runnable encapsulates run() which is an asynchronous method with no parameters and no return value
- Callable is like Runnable but returns a value
- A Future holds the *result* of an asynchronous computation

Thread Pools

- Threads are somewhat expensive to create
- Having too many concurrent threads might degrade performance

Types of Thread Pools

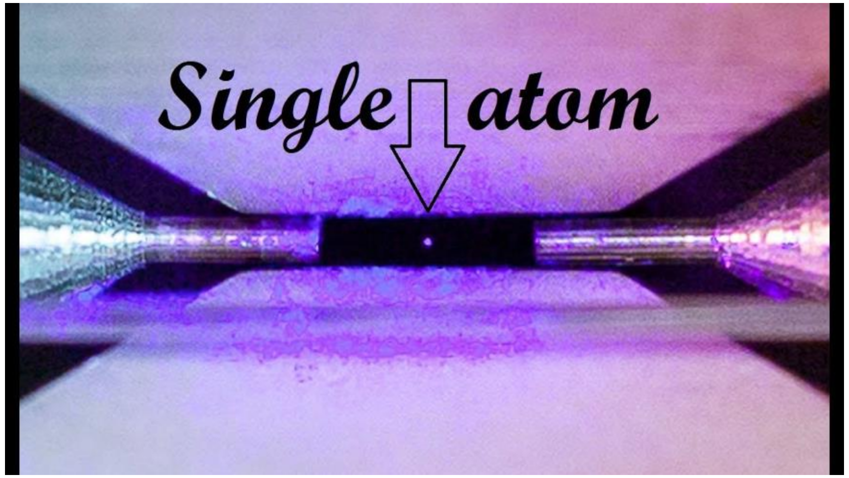
Method	Description
newCachedThreadPool	New threads are created as needed; idle threads are kept for 60 seconds.
newFixedThreadPool	The pool contains a fixed set of threads; idle threads are kept indefinitely.
newSingleThreadExecutor	A “pool” with a single thread that executes the submitted tasks sequentially (similar to the Swing event dispatch thread).
newScheduledThreadPool	A fixed-thread pool for scheduled execution; a replacement for java.util.Timer.
newSingleThreadScheduledExecutor	A single-thread “pool” for scheduled execution.

Using a Thread Pool

1. Call the static newCachedThreadPool or newFixedThreadPool method of the Executors class.
2. Call submit to submit Runnable or Callable objects.
3. If you want to be able to cancel a task, or if you submit Callable objects, hang on to the returned Future objects.
4. Call shutdown when you no longer want to submit any tasks.

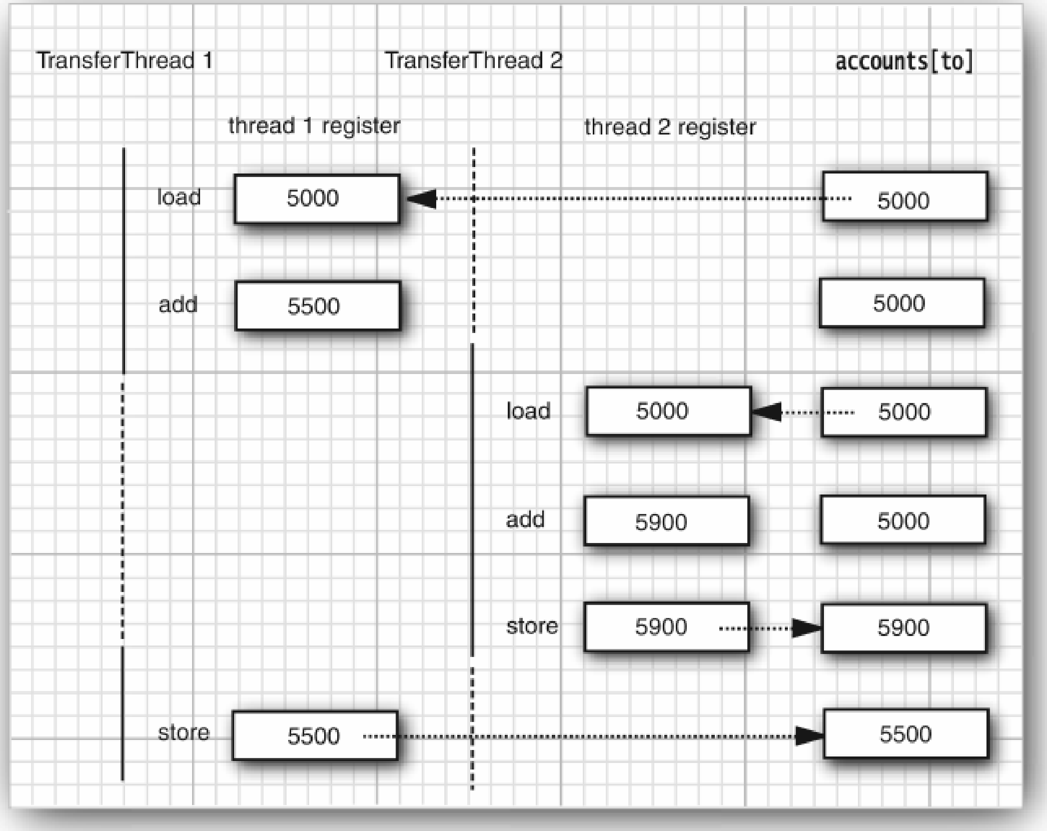
Atomicity

- Atomic means an operation that appears to be instantaneous from the perspective of all other threads
- Example:
  accounts[to] += amount;
- In reality, might not be atomic:
  1. Load accounts[to] into a register.
  2. Add amount.
  3. Move the result back to accounts[to].



Race Conditions

- Two threads have access to the same object and each calls a method that modifies the state of the object



Locks I

- A lock allows only one thread at a time can enter the critical section
- ReentrantLock

```
myLock.lock(); // a ReentrantLock object
try {
  critical section
}
finally {
  // make sure the lock is unlocked even if
  // an exception is thrown
  myLock.unlock();
}
```

Locks II

- Intrinsic lock (*synchronized* keyword)
  - Method

```
public synchronized void method() {
  method body
}
```

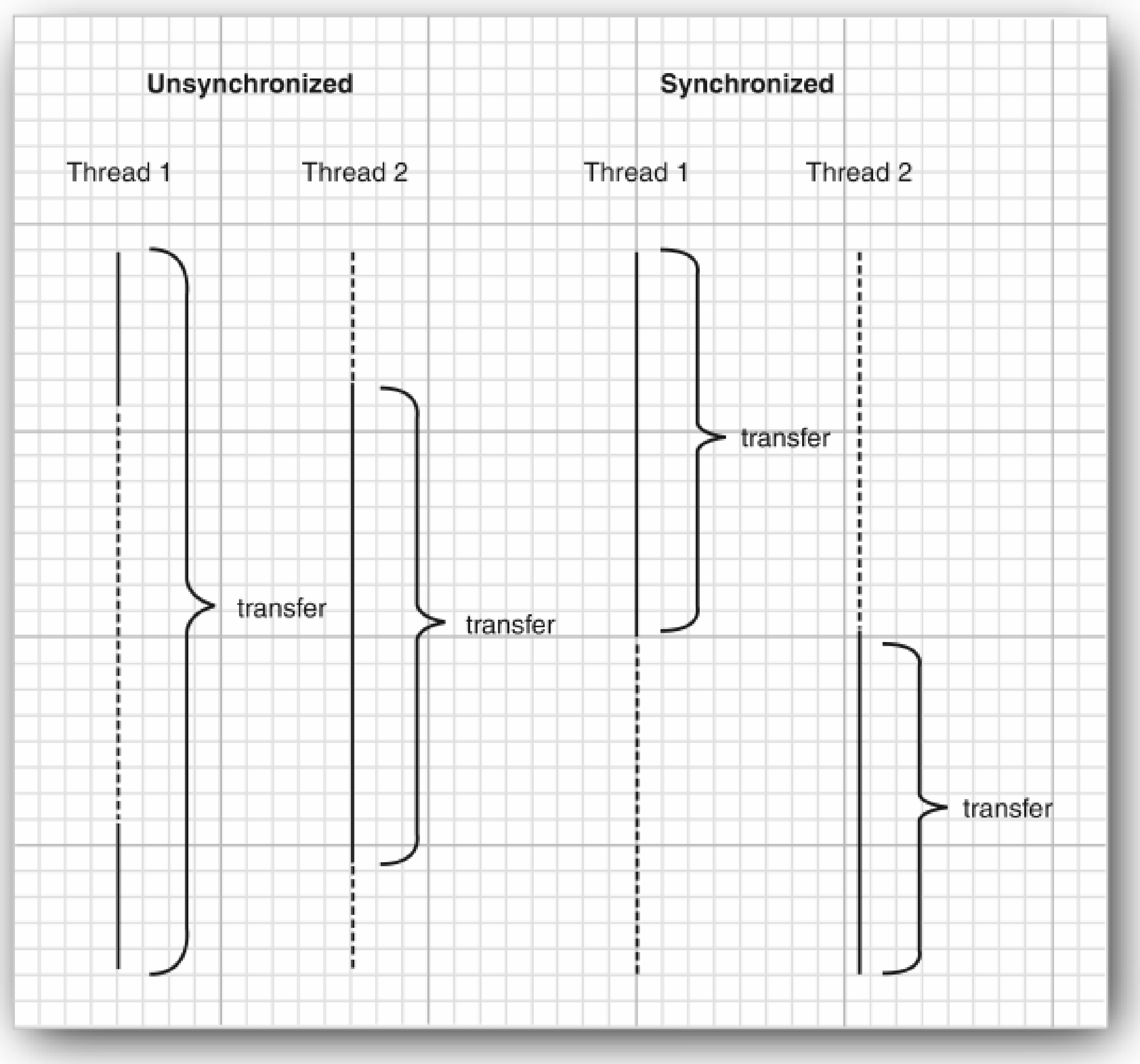
Equivalent to

```
public void method() {
this.intrinsicLock.lock();
try {
  method body
}
finally { this.intrinsicLock.unlock(); }
}
```

  - Block

```
synchronized (object) {
  block body
}
```

Locks III


Unsynchronized
Synchronized
Thread 1
Thread 2
Thread 1
Thread 2
transfer
transfer
transfer
transfer

//TODO before next lecture:

- Final Project team formation due on 3/19 at 11:59 pm EDT. Teams must be declared on Submitty.