Com S 430
Review for midterm exam

Concurrency fundamentals

Be able to recognize correct/incorrect use of synchronization locks for mutual exclusion and when/where locking is needed (shared + mutable state variables)
Fully synchronized objects (aka the "monitor" pattern)
Understand/apply the fundamental rule of locking (e.g. see Goetz, p. 28)
Class-level locks for static data, locking instance locks from inner classes
The problem of memory visibility, reordering of memory operations, fundamental rule of visibility (see Goetz, p. 37), using synchronization to ensure visibility
Semantics of wait/notify/notifyAll, the fundamental rules of condition waiting (e.g. see p. 301)
Client-side locking for traversals, why it is needed
Immutability (particularly as defined for Java, and the initialization guarantees for final fields)
Proper construction, safe publication guidelines (e.g. see p. 52)
Confinement concepts (method, instance, thread)

Using threads and concurrency utilities

Using an auxiliary thread to execute a task asynchronously
Alternatives to starting threads explicitly – using ad hoc thread pools
Confinement rules for UIs
Know how to use an auxiliary thread in a UI and provide safe updates using invokeLater
Be able to implement a simple promise-style Future, know how to use a Future as in the AST cache example, know how to use FutureTask and the Callable interface

References

Relevant sections of Goetz' book - although we covered these topics in class, the sections below are generally concise, clear, and accurate explanations.
2.1 - 2.4 - safety and locking
3.1, 3.2 - visibility
3.2.1 - safe construction
3.3 - confinement
3.4 - immutability
3.5 - safe publication
4.1, 4.2 - full synchronization (instance confinement)
4.4.1 - client-side locking
5.1, 5.2 - synchronized collections
5.6 - a result cache example (similar to the AST example in homework)
6.1, 6.2 - Executor interface and simple thread pools
9.1, 9.2, 9.3 - threads in UI frameworks
14.1, 14.2 - using wait/notify
16.1 - background on the Java Memory Model
16.3 - initialization guarantees for final fields

Problems for review

I would first suggest that you review homeworks 1 and 2 and re-do as necessary for practice. Make sure all issues are clear to you and that you can reproduce the solutions correctly. Talk to me if you need help!

More exercises to try

1) In the examples/swing_client directory there is a class SwingClientUnfinished in which the event thread is blocked during the network access. Without looking at the solutions (the other two version in that directory), try fixing so it uses a worker thread to do the network access.

2) Suppose you have a library (no source code) containing the following types. Assume that whatever it means to do "the thing", doing so produces a String result and may throw a ThingException.

```java
public interface IAsyncCallback
{
    void setResult(String result);
    void setException(ThingException e);
}

public class ThingException extends Exception
{ /* ...the usual stuff... */ }

public class Thing1
{
    // invokes appropriate callback method after doing the thing
    public void doTheThingAsync(String[] args, IAsyncCallback cb)
    { /* ...details not shown... */ }
}

Extends the class Thing1 as follows by adding a method that returns immediately with a Future for the result.

```java
public class BetterThing extends Thing1
{
    public Future<String> doTheThingAsync(String[] args)
    { /* TODO */ }
}
```

3) Suppose you have the IAsyncResult and the ThingException types as above, but that instead of Thing1 your library has the class Thing2:

```java
public class Thing2
{
```
public Future<String> doTheThingAsync(String[] args) {
    /* ...details not shown... */
}

Extend the class `Thing2` as follows by adding a method that returns a result or exception via a callback after doing the thing:

```java
public class AlsoAGoodThing extends Thing2 {
    public void doTheThingAsync(String[] args, IAsyncCallback cb) {
        /* TODO */
    }
}
```

4) When starting Swing applications we typically use this funny idiom in the `main()` method:

```java
Runnable r = new Runnable() {
    public void run() {
        createAndShow();
    }
};
SwingUtilities.invokeLater(r);
```

where `createAndShow()` is a helper method that constructs the GUI components. Explain what might go wrong if we allow the main thread to directly construct the GUI components.

5) In the examples/bounded_buffer directory there is a file `BufferWithError.java`. Give a concrete scenario in which this code fails.

6) In the examples/bounded_buffer directory there is a file `Buffer1.java`. This is a correct implementation of a bounded blocking queue. Suppose we add the method

```java
public T peek() {
    if (count == 0) throw new NoSuchElementException();
    return elements[getIndex];
}
```

a) Describe two distinct ways in which this code fails.

b) What if the variable `count` was declared `volatile`?

7) Use `wait` and `notifyAll` to implement a countdown latch including the methods `await()` and `countDown()` as described in `java.util.concurrent.CountDownLatch`