Last Updated: 2/11/15

Topic Name: Safe Parallel Threading

Brief Description:
Threads have always been a mystery to me ever since my first java class however, this time I think I've finally got it. I had to spend about 8 hours studying and watching tutorials and practice with examples along with reading a couple chapters in my java book (Head First Java) to understand what was going on. This time I've written a couple examples of threads, one using locks to make it safe and the other without in order to show the difference when I present it to my team tonight. A more detailed description of what I learned exactly is below.

Teaching Description:

Safe Parallel Threading

Threads, Runnable, and Locks

Thread

What is a thread? "A thread is an object that allows us to have *multiple threads of execution* running concurrently" – Oracle

Each time you run an application it has a **Main Thread** that executes a *stack* of instructions.
Think of each thread as being like a separate person on a team. Each person may work on the team project at the same time the other teammates are working so that the project may be completed faster. Each thread (or person) has it's *own* thread stack.

Runnable

What is **Runnable?** Runnable is an object that has a run method containing the stack that a thread wants to run.

"Runnable is to a Thread what a job is to a worker" – Head First Java

The runnable object must be passed to a thread so that the thread may access the "job" that it needs to preform.
There are three main states for a Runnable object: Runnable, Running, and Blocked.

Thread Scheduler

This controls which thread gets to take the floor and be the active thread. Not all schedulers will work the same way on every machine though so we need to rely on the **sleep()** method of a thread to allow other threads time to have a turn.

Thread Locks

The purpose for locks in threading arises when two threads are accessing the same methods data and changing it at the same time. Let's consider two people A and B where A and B both download a word document at the same time. They each make changes and edit it differently. Then A decides to upload the file back to where it

came from with the changes he made. Next B decides to upload his file. When B uploads his file it overwrites all the work that A has done and it is lost forever.
This is where a **lock** comes in handy. A lock allows **only one thread** (or in our example one person) to hold **the one and only key** to download, change, and then upload the document at a time.
In Java, every object has a lock – but for the most part they are all left "unlocked" so any thread may access them.
In order to use a lock you must put the keyword **synchronized** in front of your method. This is how Java knows that the lock and key will be actively used.
One important thing to note is that the **key** is owned by the **object** and not the individual method that is locked.
Or in other words - even though we place the keyword synchronized in front of methods it doesn't mean that we can have two synchronized methods in a single object with each of them owning their own individual key therefore letting different threads access each of the methods separately at the same time.
What it really means is that if a thread holds the key, that is the only key to access ANY of synchronized methods contained in that object. In essence each object has only one key to unlock all the synchronized methods within it, and can be held by only one thread at a time.

Deadlock
One of the main problems with locks is that a thread will not pick up another key before dropping the key it holds currently. So if you have two objects, and two threads, and each thread is holding a key to a different object but both threads want to gain access to the object that the other thread is holding they will both wait indefinitely for the other thread to drop their key first before they drop their own.
This is called a deadlock.

A Problem with Locking
One problem that can occur with locking is that it forces threads to wait in line to access locked objects. This means that our program will not run as fast because we are loosing some of our concurrency.

Teaching Examples:
Videos
Team Teaching Video from 5:00-16:00
http://youtu.be/oFND1NeVr_U?t=5m

Sharing Video
http://youtu.be/qP3Y7DFlcVk

Files to View:
Safe Parallel Processing/BankAccountProblem
Safe Parallel Processing/ BankAccountProblemWithLocks