Assignment 3 Part 1

(may be done by a team of at most two students)

Assigned: Friday, October 23

Due: Friday, Nov 6, 11:59 pm **(Parts 1 and 2)**

Part 1: Thread Safe Trees

Refer to file `ThreadSafeTree.java` whose `main` method creates and starts five concurrent threads, each of which inserts five random integers into a tree, and finally prints out all values in the tree. The code as given invokes the standard `Tree.insert` method, which is *not thread-safe*, meaning that it is not suited for concurrent insertion of values into the tree, as values will be dropped from the tree due to the lack of any synchronization – as can be observed from the object diagram.

A preliminary solution to this problem is to declare `insert` as a `synchronized` method. Doing so will solve the problem of dropped values, but this solution is not desirable because it sacrifices concurrency – essentially, all values are now inserted sequentially into the tree. You might try this out as a preliminary step.

Your task in Part 1 is to write a subclass `SafeTree` of class `Tree` with a *thread-safe* definition of `insert(n)`, i.e., it preserves the basic logic of `Tree.insert`, but permits concurrent insertion of values into the tree, with no dropped values. Concurrent insertion into disjoint subtrees as well as concurrent insertion at different nodes along any path from the root to a leaf should be supported. To meet these objectives:

1. Do not declare `SafeTree.insert(n)` as a synchronized method.

2. Define two synchronized methods in class `SafeTree`, called `lock()` and `unlock()`, using which `SafeTree.insert(n)` can ensure that one thread at a time is accessing any given `SafeTree` object, but other `SafeTree` objects can be accessed concurrently by other threads.

3. Call the `lock()` and `unlock()` methods from `SafeTree.insert(n)` in such a way that any `SafeTree` object is locked for as short a duration as possible.

4. Define `lock()` and `unlock()` using Java's `wait-notify` constructs. Their definitions are similar to other `wait-notify` examples discussed in the lectures.

Run `ThreadSafeTree.java` to completion after replacing the instruction '`new Tree(5000)`' by '`new SafeTree(5000)`' in `ThreadSafeTree.main`. Proceed as follows:

1. Check the JIVE object diagram to make sure that it does not have any dropped values, and check the console output to make sure that the printed values are in ascending order.

2. Bring up the JIVE Search window, choose the *Object Created* option, enter `SafeTree` for the class name, and press *Search*. There should be 31 entries in the Search Results window for the given test case.

3. Step through the search results one by one, observing the object diagram (using the '*Objects*' option) at each step, until you locate at an object diagram showing *maximal concurrency*, i.e., where there is a maximum number of active threads in disjoint subtrees and also a maximum number of active threads at different nodes along a path in the tree. There could be more than one such diagram; choose any object diagram with *maximal concurrency*.

4. Save the chosen object diagram from step 3 in a file called `A3_part1.png`.

What to Submit. Prepare a top-level directory named *A3_Part1_UBITId1_UBITId2* if the assignment is done by a team of two students; otherwise, name it as *A3_Part1_UBITId* if the assignment is done solo. (Order the *UBITId*s in alphabetic order, in the former case.)

In this directory, place your `ThreadSafeTree.java` and `A3_part1.png`. Compress the directory and submit the compressed file using `submit_cse410` (undergrads) or `submit_cse522` (grads). Only one submission per team is required.

End of Assignment 3 Part 1

Part 2: To be assigned

Using the JIVE Search Window

Click in the source code window and do Ctrl-h in order to bring up the JIVE Search Window. Alternatively, from Eclipse's Search menu, select the first entry, also called Search. Then select the JIVE Search tab from the menu of tabs presented.

For Part 1, choose the *Object Created* query option, enter SafeTree for the class name, and press *Search*. There should be 31 entries in the Search Results window for the given test case.

The search results are shown in under the *Search tab* located near the Console. If the results are not being shown as a table, use the inverted triangle menu located third from the right, and choose the *Show as List* option.

Using the two solid yellow arrows (called *Show Next Match* and *Show Previous Match*) in order to cycle through the results. Observe the object diagram (in the *Objects* mode) and save the diagram with maximal concurrency.