# Assignment 4

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

Assigned: Thursday, October 26
Due: Wednesday, November 7 (11:59 pm)

***Part 1: Concurrent Mutator-Collector Simulator***

Java uses a variety of garbage-collection techniques, one of which is the Concurrent Mark Sweep (CMS) Collector. Lecture 17 describes the two main problems of concurrent marking. In this part of the assignment you are to program a solution to these problems for the specific example where the mutator performs sequential insert and delete operations on a binary search tree (BST).

Posted on `Piazza:Resources → Assignments` is a file `ConcurrentMarking.java` consisting of two thread classes, `Mutator` and `Collector`, and a `main` method which creates and starts these two threads concurrently. The class `Mutator` inserts a sequence of values into a BST and then deletes some of the values from the BST. The `Collector` class defines the `mark` method which performs a depth-first traversal of the BST and sets the `mark_bit` fields of all nodes to `true`. The definitions of the BST class `Tree` and the `insert` and `delete` methods are as in Assignment 1 and are given in the posted file.

Lecture 17 slide 13 clarifies the two types of incorrect markings: (i) the `mark_bit` fields of some of the un-deleted nodes of the tree are `false`; and (ii) the `mark_bit` fields of some deleted nodes are `true`. The cause of both problems is that the collector visited these nodes before some of their fields got modified. Solve the first problem by extending `Tree.insert` so that the `mark_bit` fields for these nodes are set to `true`, and add the node values to the sorted list, `re_mark_true`, in class `Collector`. Solve the second problem by extending `Tree.case1` and `Tree.case2` so that the `mark_bit` fields for these nodes are set to `false`, and add the node values to the sorted list, `re_mark_true`, in class `Collector`.

***What to Code.*** The amount of your coding is very small and it is very systematic. In practice, the compiler emits a small amount of code before every write operation to a variable that could hold an object reference. This code would alert the garbage collector as to where re-marking is necessary. Your Java code extensions should effectively carry out the required re-marking. The lists `re_mark_true` and `re_mark_false` help us see what re-markings took place.

After making the required extensions, run your revised `ConcurrentMarking.java` under JIVE and proceed as follows:

1. Save the object diagram (stacked with tables) in a file called `A4_obj.png`.

2. Check the JIVE object diagram to make sure that the `mark_bit` values are correct for the (un-deleted) nodes of the tree as well as for the deleted nodes of the tree.

3. For the given test case, the values 50, 90, 100 and 150 involve a case3 delete, hence the nodes containing these values are replaced, respectively, by values 55, 95, 133, and 160. Thus, the values 55, 95, 133, and 160 should appear in the list of deleted nodes at the bottom of the object diagram (rather than 50, 90, 100, and 150).

4. Perform a JIVE Search with `mark_bit == false`, and check that all the objects in the Search Results are present among deleted objects at the bottom of the Object Diagram and also that the values in the objects appear on the Console under 'Nodes deleted late'. For any additional deleted Tree objects in the Object Diagram, perform a JIVE Search for their `mark_bit` and check that are 0 assignments to the `mark_bit`.

***What to Submit*.** Prepare a top-level directory named *A4_Part1_UBITId1_UBITId2* if the assignment is done by a team of two students; otherwise, name it as *A4_Part1_UBITId* if the assignment is done solo. (Order the *UBITId*s in alphabetic order, in the former case.) In this directory, place your `ConcurrentMarking.java` and `A4_obj.png`. Compress the directory and submit the compressed file using the `submit_cse522` command. Only one submission per team is required.

***Part 2: Design Patterns***

To Be Assigned.

**End of Assignment 4**