```java
import java.util.concurrent.locks.ReentrantLock;

public class Node {

        public static int MAX_LEVEL = 10;
        final ReentrantLock Lock = new ReentrantLock();
        final int key;
        final Node [] next ;
        volatile boolean marked = false;
        volatile boolean fullyLinked = false;
        public int toplevel;

        public Node(int key){   //sentinel constructor
                this.key = key;
                next = new Node[MAX_LEVEL+1];
                this.toplevel = MAX_LEVEL;
        }

        public Node (int key, int level){//non sentinel constructor
                this.key = key;
                next = new Node[level+1];
                this.toplevel = level;
        }

        public void lock(){
                Lock.lock();
        }

        public void unlock(){
                Lock.unlock();
        }
}
```

```java
import java.util.Random;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class TestConc implements Runnable{

        static LazyList l;
        public static Lock printLock = new ReentrantLock();
        public static void main(String[] args){
                l = new LazyList();
                for(int i=0;i<3;i++)
                        new Thread(new TestConc()).start();
        }
        public void run(){
                Random rnd = new Random();
                for(int loop = 0; loop < 10; loop++){
                        int seed = rnd.nextInt(50);
                        if(seed%3 == 0){
                                if(l.contains(seed) == false){
                                        l.add(seed);
                                }
                                else{
                                        printLock.lock();
                                        System.out.println(seed+ " Already exists");
                                        printLock.unlock();
                                }

                        }
                        else if(seed%3 == 1){
                                if (l.contains(seed) != false)
                                {
                                        l.remove(seed);
                                }
                                else
                                {
                                        printLock.lock();
                                        System.out.println(seed + " Not found!");
                                        printLock.unlock();
                                }
                        }
                        else{
                                printLock.lock();
                                System.out.println("Printing the skipList");
                                l.printList();
                                System.out.println("Printed the skipList");
                                printLock.unlock();
                        }

        }
}}
```

```java
import java.util.Random;
public class LazyList {
        public static final int MAX_LEVEL = 10;
        final Node head = new Node(Integer.MIN_VALUE);
        final Node tail = new Node(Integer.MAX_VALUE);
        public LazyList(){
                for(int i =0 ; i<head.next.length; i++){
                        head.next[i] = tail;
                }
        }

        public int find(int key, Node[] preds, Node[] succs){
                //iterates from the top of the list to the bottom most level
                //locates the node
                //fills preds and succs array
                Node curr,pred;
                pred = head;
                int lFound = -1;
                for(int level= MAX_LEVEL; level >= 0; level--){
                        curr = pred.next[level];
                        while(curr.key < key){
                                pred = curr;
                                curr = pred.next[level];
                        }
                        if (lFound == -1 && curr.key == key)
                                lFound = level;

                        preds[level] = pred;
                        succs[level] = curr;
                }

                return lFound;
        }

        public boolean add(int key){
                Random rnd = new Random();
                //get the height by coin flip
                int topLevel=0; //stores the topmost level where this node is added
                while( rnd.nextInt(2)%2 == 1)   //till you keep getting heads
                        topLevel+=1;

                topLevel = Math.min(topLevel, MAX_LEVEL);

                Node [] preds = new Node[MAX_LEVEL+1];
                Node [] succs = new Node[MAX_LEVEL+1];
                Node pred,succ ;
                while(true){
                        int lFound = find(key, preds, succs);
                        if (lFound != -1){//node is found
                                Node nodeFound = succs[0];
                                if (!nodeFound.marked){
                                        while(nodeFound.fullyLinked != true){
                                        //wait till fully linked
                                        }
                                        return false;
                                }
                                continue;
                        }
                        //if not found
                        //* lock all preds
                        boolean valid = true;
                        int highestLocked=0;
                        try{
                                pred = head;
                                for(int level = 0; valid && (level <= topLevel) ; level++){
                                        pred = preds[level];
                                        succ = succs[level];
                                        pred.lock();

                                        highestLocked = level;

                                        //* validate all preds
                                        valid = !pred.marked && !succ.marked && (pred.next[level] == succ);
                                }
                                if (!valid)//valid criteria failed
                                        continue;

                                Node newNode = new Node(key, topLevel);
                                //assign pointers
                                for(int level = 0; level <= topLevel; level++){
                                        newNode.next[level] = succs[level];
                                }
                                for(int level = 0 ; level <= topLevel; level++){
                                        preds[level].next[level] = newNode;
                                }
                                //* set linearization point
                                newNode.fullyLinked = true;
                                return true;
                        }
                        finally{
                                for(int level = 0; level <= highestLocked; level++){
                                //      System.out.println("@level: "+ level);
                                        pred = preds[level];
                                        pred.unlock();
                                }
                        }
                }
        }

        public boolean remove(int key){
                Node[] preds = new Node[MAX_LEVEL+1];
                Node[] succs = new Node[MAX_LEVEL+1];
                Node victim = null;
                boolean isMarked = false;
                int topLevel = -1;

                while(true){
                        int lFound = find(key, preds, succs);
                        if (lFound != -1)//if node is found
                                victim = succs[lFound];
                        if(isMarked | (lFound != -1 && (victim.fullyLinked && victim.toplevel == lFound && !victim.marked) )){

                                if(!isMarked){
                                        topLevel = victim.toplevel;
                                        victim.lock();
                                        if(victim.marked){
                                                victim.unlock();
                                                return false;
                                        }

                                        victim.marked= true;
                                        isMarked = true;
                                }

                                int highestLocked = -1;
                                try{

                                        Node pred;
                                        boolean valid = true;
                                        for(int level = 0; valid && (level <= topLevel); level++){
                                                pred = preds[level];
                                                pred.lock();
                                                highestLocked = level;
                                                valid = !pred.marked && pred.next[level] == victim;
                                        }
                                        if(!valid)
                                                continue;

                                        for(int level = topLevel; level >= 0; level--){
                                                preds[level].next[level] = victim.next[level];
                                        }
                                        victim.unlock();
                                        return true;
                                }
                                finally{
                                        for(int i =0 ; i<= highestLocked; i++)
                                                preds[i].unlock();
                                }
                        }
                        else
                                return false;
                }
        }

        public boolean contains(int key){
                Node [] preds = new Node[MAX_LEVEL+1];
                Node [] succs = new Node[MAX_LEVEL+1];
                int lFound = find(key, preds,succs);
                return (lFound != -1 && succs[lFound].fullyLinked && !succs[lFound].marked);
        }

        void printList(){
                Node curr;
                for(int level = MAX_LEVEL; level >=0 ; level--){
                        curr = head;
                        while(curr.next[level].key < tail.key){
                                System.out.print(curr.next[level].key + " ");
                                curr = curr.next[level];
                        }
                        System.out.println();
                }
                return;
        }
}
```