```
1  public class WFSnapshot<T> implements Snapshot<T> {
2    private StampedSnap<T>[] a_table; // array of atomic MRSW registers
3    public WFSnapshot(int capacity, T init) {
4      a_table = (StampedSnap<T>[]) new StampedSnap[capacity];
5      for (int i = 0; i < a_table.length; i++) {
6        a_table[i] = new StampedSnap<T>(init);
7      }
8    }
9    private StampedSnap<T>[] collect() {
10     StampedSnap<T>[] copy = (StampedSnap<T>[]) new StampedSnap[a_table.length];
11     for (int j = 0; j < a_table.length; j++)
12       copy[j] = a_table[j];
13     return copy;
14   }
15   public void update(T value) {
16     int me = ThreadID.get();
17     T[] snap = scan();
18     StampedSnap<T> oldValue = a_table[me];
19     StampedSnap<T> newValue = new StampedSnap<T>(oldValue.stamp+1, value, snap);
20     a_table[me] = newValue;
21   }
22   public T[] scan() {
23     StampedSnap<T>[] oldCopy, newCopy;
24     boolean[] moved = new boolean[a_table.length]; // initially all false
25     oldCopy = collect();
26     collect: while (true) {
27       newCopy = collect();
28       for (int j = 0; j < a_table.length; j++) {
29         if (oldCopy[j].stamp != newCopy[j].stamp) {
30           if (moved[j]) {
31             return newCopy[j].snap;
32           } else {
33             moved[j] = true;
34             oldCopy = newCopy;
35             continue collect;
36           }
37         }
38       }
39       T[] result = (T[]) new Object[a_table.length];
40       for (int j = 0; j < a_table.length; j++)
41         result[j] = newCopy[j].value;
42       return result;
43     }
44   }
45 }
```

FIGURE 4.20 Single-writer atomic snapshot class.