```
37    public T removeMin() {
38      heapLock.lock();
39      int bottom = --next;
40      heap[ROOT].lock();
41      heap[bottom].lock();
42      heapLock.unlock();
43      T item = heap[ROOT].item;
44      heap[ROOT].tag = Status.EMPTY;
45      heap[ROOT].owner = NO_ONE;
46      swap(bottom, ROOT);
47      heap[bottom].unlock();
48      if (heap[ROOT].tag == Status.EMPTY) {
49        heap[ROOT].unlock();
50        return item;
51      }
52      heap[ROOT].tag = Status.AVAILABLE;
53      int child = 0;
54      int parent = ROOT;
55      while (parent < heap.length / 2) {
56        int left = parent * 2;
57        int right = (parent * 2) + 1;
58        heap[left].lock();
59        heap[right].lock();
60        if (heap[left].tag == Status.EMPTY) {
61          heap[right].unlock();
62          heap[left].unlock();
63          break;
64        } else if (heap[right].tag == Status.EMPTY || heap[left].score < heap[right].score) {
65          heap[right].unlock();
66          child = left;
67        } else {
68          heap[left].unlock();
69          child = right;
70        }
71        if (heap[child].score < heap[parent].score && heap[child].tag != Status.EMPTY) {
72          swap(parent, child);
73          heap[parent].unlock();
74          parent = child;
75        } else {
76          heap[child].unlock();
77          break;
78        }
79      }
80      heap[parent].unlock();
81      return item;
82    }
```

FIGURE 15.10 The FineGrainedHeap class: the removeMin() method.