QUIZIZZ **Worksheets**

Session 12 PProg

Total questions: 9

Worksheet time: 16mins

Instructor name: Mr. Jonas Wetzel

Name

Class

Date

1.

```
public boolean add(T item) { 5 references
  int key = item.hashCode();
  while (true) {
    Node pred = this.head;
    Node curr = pred.next;
    // traverse without locks
    while (curr.key < key) {
      pred = curr;
      curr = curr.next;
    }
    // when arrived, lock predecessor and successor
    pred.lock();
    curr.lock();
    try {
      // validate if still reachable
      if (validate(pred, curr)) {
        if (curr.key == key) { // already present
          return false;
        } else {
          Node entry = new Node(item);
          pred.next = entry;
          entry.next = curr;
          return true;
        }
      }
    } finally { // always unlock
      curr.unlock();
      pred.unlock();
    }
  }
}
```

OPTIMISTIC implementation of ADD() method. Are bad interleavings possible?

a) No, code is correct.

b) Yes, because curr and pred are being unlocked in wrong order.

c) Yes, because of an error in the code logic.

d) Yes, because pred.next is set before entry.next.

e) Yes, because list is traversed without locks.

2. What are advantages of lazy vs. optimistic synchronization?

a) Lazy sync. requires no lock acquisitions.

b) add()/remove() require less lock acquisitions

c) add()/remove() require less list traversals

d) remove() is in O(1)

3.

```
synchronized public void push(Long item) {
   top = new Node(item, top);
}

synchronized public Long pop() {
   if (top == null)
      return null;
   Long item = top.item;
   top = top.next;
   return item;
}
```

A stack with these push() and pop() operations is non-blocking.

a) True

b) False

4. Wait-Free implies Starvation-Free

   a) True
   b) False

5. Lock-Free implies starvation-free.

   a) True
   b) False

6. Wait-Free implies Lock-Free

   a) True
   b) False

7. Lock-free implies deadlock-free.

   a) True
   b) False

8.
```
private int factor = 1;
private final AtomicBoolean full = new AtomicBoolean(false);

public void Update1(int multiplier) {
  while (!full.compareAndSet(false, true));
  factor *= multiplier; //CS
  full.set(false);
}
```

   The algorithm is lock-free. No assumptions about CS.

   a) True
   b) False
   c) It is incorrect due to a data race.

9.
```
private final AtomicInteger factor = new AtomicInteger(1);

boolean Update2(float multiplier, int retries) {
  while (retries >= 0) {
    int val = factor.get();
    int res = val * multiplier;
    if (factor.compareAndSet(val, res)) {
      return true;
    }
    --retries;
  }
  return false;
}
```

   The code is... (select all that apply)

   a) Deadlock-Free
   b) Lock-Free
   c) Starvation-Free
   d) Wait-Free
   e) Incorrect because of the ABA problem.

Answer Keys

1. d) Yes, because pred.next is set before entry.next.
2. c) add()/remove() require less list traversals
3. b) False
4. a) True
5. b) False
6. a) True
7. a) True
8. b) False
9. a) Deadlock-Free, b) Lock-Free, c) Starvation-Free, d) Wa