```java
class VolatileExample {
  int x = 0;
  volatile boolean v = false;
  public void writer() {
    x = 42;
    v = true;
  }
  public void reader() {
    if (v == true) {
      int y = 100/x;
    }
  }
}
```

Figure 3.16 Volatile field example from Exercise 28.

Exercise 29. Is the following property equivalent to saying that object x is wait-free?

> For every infinite history H of x, every thread that takes an infinite number of steps in H completes an infinite number of method calls.

Exercise 30. Is the following property equivalent to saying that object x is lock-free?

> For every infinite history H of x, an infinite number of method calls are completed.

Exercise 31. Consider the following rather unusual implementation of a method m. In every history, the i^{th} time a thread calls m, the call returns after 2^i steps. Is this method wait-free, bounded wait-free, or neither?

Exercise 32. This exercise examines a queue implementation (Fig. 3.17) whose `enq()` method does not have a linearization point.

The queue stores its items in an `items` array, which for simplicity we will assume is unbounded. The `tail` field is an `AtomicInteger`, initially zero. The `enq()` method reserves a slot by incrementing `tail`, and then stores the item at that location. Note that these two steps are not atomic: there is an interval after `tail` has been incremented but before the item has been stored in the array.

The `deq()` method reads the value of `tail`, and then traverses the array in ascending order from slot zero to the tail. For each slot, it swaps *null* with the current contents, returning the first non-*null* item it finds. If all slots are *null*, the procedure is restarted.

Give an example execution showing that the linearization point for `enq()` cannot occur at Line 15.

Hint: give an execution where two `enq()` calls are not linearized in the order they execute Line 15.

```
1  public class HWQueue<T> {
2    AtomicReference<T>[] items;
3    AtomicInteger tail;
4    static final int CAPACITY = 1024;
5
6    public HWQueue() {
7      items =(AtomicReference<T>[])Array.newInstance(AtomicReference.class,
8          CAPACITY);
9      for (int i = 0; i < items.length; i++) {
10       items[i] = new AtomicReference<T>(null);
11     }
12     tail = new AtomicInteger(0);
13   }
14   public void enq(T x) {
15     int i = tail.getAndIncrement();
16     items[i].set(x);
17   }
18   public T deq() {
19     while (true) {
20       int range = tail.get();
21       for (int i = 0; i < range; i++) {
22         T value = items[i].getAndSet(null);
23         if (value != null) {
24           return value;
25         }
26       }
27     }
28   }
29 }
```

Figure 3.17 Herlihy/Wing queue.

Give another example execution showing that the linearization point for `enq()` cannot occur at Line 16.

Since these are the only two memory accesses in `enq()`, we must conclude that `enq()` has no single linearization point. Does this mean `enq()` is not linearizable?

Exercise 33. Prove that sequential consistency is nonblocking.