Answers - 2016

Question 1

1.1

```
Sum is 1599166.000000 and should be 2000000.000000
Sum is 1814240.000000 and should be 2000000.000000
Sum is 1858673.000000 and should be 2000000.000000
Sum is 1858673.000000 and should be 2000000.000000
Sum is 1680947.000000 and should be 2000000.000000
```

The class does not seem to be thread-safe, since the results aren't 2000000.000000.

1.2

Since the locking happens on both the instance of TestLocking0 and the static class of TestLocking0, the results are not what we expect. We get race conditions since the instance is locked in one thread, but the other thread can still continue working, since it is locking on the class object.

1.3

We would not lock on both the class object and the instance of the class. We would instead lock only on the instance in btoh threads, so we guarantee thread safety using locking. The changes can be seen below:

```
class Mystery {
  private static double sum = 0;

  public synchronized void addStatic(double x) {
    sum += x;
  }

  public synchronized void addInstance(double x) {
    sum += x;
  }

  public synchronized double sum() {
    return sum;
  }
}
```

Results:

```
Sum is 2000000.000000 and should be 2000000.000000
Sum is 2000000.000000 and should be 2000000.000000
Sum is 2000000.000000 and should be 2000000.000000
Sum is 2000000.000000 and should be 2000000.000000
Sum is 2000000.000000 and should be 2000000.000000
```

Question 2

2.1

We would make `get`, `add`, `set`, `toString` synchronized. This is the simplest way to make the class thread-safe in our opinion. We don't need to make `size` synchronized, since the `size` member on the class is read-only in this implementation and therefore cannot lead to race conditions.

2.2

This simple thread-safe implementation would not scale very well, since we lock the entire array of items whenever a thread calls either of `get`, `add`, `set`, `toString`.

2.3

Threads will wait for each other while trying to do each single operation, but this does not achieve thread-safety since it is possible to insert into the list while fetching an object out of the list, which will lead to race-conditions. Visibility is broken in this implementation since it is possible for one thread to insert while another either fetches an object or tries to get the size. These actions can happen concurrently, but the thread getting size or an object in the list might not see the update.

2.4

Using striping where we lock n sections of the inner array on all operations would give us better performance and guarantee visibility.