Exercise 1.1

1. After removing the synchronized, the result is always mismatching.
2. This kind of result happened because two threads are sharing the state at the same time and in this kind of cases the threads deals with the stale data which causes the program to give unwanted results. This is a bad solution.
3. No, it does not make any difference by using such kind of forms. The result is worse in all cases.
4. Yes, of course the methods have to be synchronized in order to give the right result because both the threads are changing the state and the two threads will always see the stale data.
5. Only when both the methods are synchronized, it produces the correct result because of we can obtain mutual exclusion and only one thread can execute the code at a time.

Exercise 1.2

1. This kind of scenario can happen due to the lack of mutual exclusion and one thread might call the print function in the middle of the operation which causes to bars or dashes to print at the same time.
2. It works as expected after declaring the print method synchronized because after synchronization only one thread can access the print method at once and no other thread can interfere in the middle.

Exercise 1.3

1. Yes, I observe the same problem.
2. Yes, after declaring both the functions synchronized it performs smoothly.
3. The synchronized is needed in both the methods so that the value is visible to the reader thread.
4. Volatile is enough in this case because it is only about the visibility, there is no any atomic operation involved in it.

Exercise 1.4

1. The class seems to be not thread safe.
2. The class is not thread safe because it takes it as a two different lock and to be thread safe there should be same lock. Static synchronized means holding lock on the the class's Class object whereas instance method to be synchronized means holding lock on that class's object itself.
3. If both are locked with Mystery.class we can make it thread safe.

Exercise 1.5

1. In order to maintain the thread safety, proper locking should be done (synchronized) where the state are being changed to obtain atomicity.
2. If a large number of threads call the get,add and set concurrently the performance is badly hurt as each thread locks the operation until it finishes which take huge amount of time.
3. No, it would not achieve thread safety as there are different locks used which will not obtain atomicity.

Exercise 1.6

1. We should synchronize the blocks where totalSize is read and written with the class literal DoubleArrayList.class.
2. Synchronize the block inside the constructor with the class literal DoubleArrayList.class. Same with the allLists() method.

Exercise 1.7

1. The methods are locked with class literal object. So, increment is locked by MysteryA.class whereas increment4 is locked by MysteryB.class and therefore there is no thread safety which results in unexpected result.
2. We can use final static Object mylock=new Object() to lock increment and increment4 methods to maintain thread safety and produce expected result.