Exercise 1.1

Question 1

If the increment() method is not synchronized then you cannot be sure of what the final value of count will be. Two threads will be able to call the method simultaneously where one of the calls overwrites the other and thus causes thread interference.

Question 2

The lower the amount of increments the less is the risk of thread interference. The software will still be wrong as it will have the risk of performing an inaccurate number of increments of count.

Question 3

No, the different ways of incrementing the count results in the same error. The statements are all compound statements (check-modify-write sequences) that must execute atomatically to be thread-safe.

Question 4

The final count is expected to be 0. If not both methods are synchronized and share the same intrinsic lock of the LongCounter object, the operations could get interleved. Though the two threads call different methods, they still act on the same mutable shared state.

Question 4

I think the explanation for the incorrect final values of each of the four examples is given above. i) The methods will be executed simultaneously and thus will cause thread interferences. 15280, 20441, -3364 ii) If only one method is synchronized it is not possible to control the lock and thus the count value is not predictable. 4701, 38383, 25257 iii) Same as above. iv) The synchonization makes them share the object's intrinsic lock which ensures atomic executions.

Exercise 1.3

Question 1

1. t1.print()
2. t2.print()
3. t1 sleeps after printing '-'
4. t2 print '-' while t1 sleeps
5. t1 print '|' after catching an interruptedException
6. t2 print '|' after sleep or when catching an interruptedException This will print ||--

Question 2

Making print() synchronized prevents thread interference so the print method will be atomic. The intrinsic lock of the Printer object can only be held by one thread and thus only one thread can execute the method at a given time. Thus the thread will still have the lock while sleeping.