# Single producer consumer problem:

Thread one -> market data producer

Thread two -> market data consumer

Efficient way to address this problem is to use a fifo buffer into which the producer inserts the data and consumer reads and removes it. It is necessary to ensure that the operations by producer/consumer are thread safe.

Data structure: A blocking queue. Consumer waits if the queue is empty till the producer inserts the data, producer waits if the queue is full till the consumer consumes the data. This can be achieved by putting consumer thread to sleep if the queue is empty and notifying it when the producer adds the data. Same logic holds for producer thread, it’ll be put to sleep if the queue is full and will be notified if the data is consumed by consumer thread. Challenge here is to ensure that the threads don’t go into deadlock state because of race conditions. It can be ensured using monitor over critical section of the code or

In java, ArrayBlockingQueue or LinkedBlockingQueue can be used which internally implements the above functionality. Above data structures work for multiple consumer-producer scenario as well because the insert/remove operations are atomic (using internal locks).

Pseudo code for one producer one consumer case:

Main() {

BlockingQueue<Data> q = new ArrayBlockingQueue<Data>();

Producer p = new Producer(q);

Consumer c = new Consumer(q);

New Thread(p).start();

New Thread( c). start();

}

## Dent shooting range:

Binary search

## SQL multiple update:

UPDATE config

SET config\_value = CASE config\_name

WHEN 'name1' THEN 'value'

WHEN 'name2' THEN 'value2'

ELSE config\_value

END

WHERE config\_name IN('name1', 'name2');

## DB normalization

https://www.sqa.org.uk/e-learning/MDBS01CD/page\_27.htm