*public class* BlockingQueue<T> {
 *private Queue*<T> queue = *new* LinkedList<>();
 *private final int* capacity;

 *public* BlockingQueue(*int* capacity) {
 *super*();
 *this*.capacity = capacity;
 }

 *public void* put(T item) *throws* InterruptedException {
 *synchronized* (queue) {
 *while* (queue.size() == capacity) {
 queue.wait();
 }
 queue.add(item);
 queue.notifyAll();
 }
 }

 *public* T take() *throws* InterruptedException {
 *synchronized* (queue) {
 *while* (queue.isEmpty()) {
 queue.wait();
 }
 T item = queue.remove();
 queue.notifyAll();
 *return* item;
 }
 }

 *public int* size() {
 *synchronized* (queue) {
 *return* queue.size();
 }
 }

 *public static void* main(String[] args) *throws* InterruptedException {
 BlockingQueue<Integer> queue = *new* BlockingQueue<>(3);
 Random random = *new* Random();
 *Runnable* p = () -> {
 *try* {
 *while* (*true*) {
 *int* element = random.nextInt(10);
 queue.put(element);
 System.*out*.println("Produced " + element + " and current size " + queue.size());
 Thread.*sleep*(random.nextInt(3) \* 1000 );
 }
 } *catch* (InterruptedException e) {
 e.printStackTrace();
 }
 };

 *Runnable* c = () -> {
 *try* {
 *while* (*true*) {
 *int* element = queue.take();
 System.*out*.println("Consumed " + element + " and current size " + queue.size());
 Thread.*sleep*(random.nextInt(3) \* 1000 );
 }
 } *catch* (InterruptedException e) {
 e.printStackTrace();
 }
 };

 Thread producerThread = *new* Thread(p);
 Thread consumerThread = *new* Thread(c);
 producerThread.start();
 consumerThread.start();

 }
}

Consumed 3 and current size 0

Produced 3 and current size 0

Produced 8 and current size 1

Produced 7 and current size 2

Consumed 8 and current size 1

Produced 5 and current size 2

Consumed 7 and current size 1

Produced 1 and current size 2

Produced 5 and current size 3

Consumed 5 and current size 2

Consumed 1 and current size 1

Produced 7 and current size 2

Consumed 5 and current size 1