[*https://raw.githubusercontent.com/vsaravanan/java22/master/src/main/java/com/saravanjs/java22/console/multithreading/semaphore/ProducerConsumerExample.java*](https://raw.githubusercontent.com/vsaravanan/java22/master/src/main/java/com/saravanjs/java22/console/multithreading/semaphore/ProducerConsumerExample.java)

*class* SharedBuffer {
 *private final Queue*<Integer> buffer;
 *private final int* maxSize;
 *private final* Semaphore items; *// Semaphore counting available items
 private final* Semaphore spaces; *// Semaphore counting available spaces
 private final* Semaphore mutex; *// Binary semaphore for mutual exclusion

 public* SharedBuffer(*int* maxSize) {
 *this*.buffer = *new* LinkedList<>();
 *this*.maxSize = maxSize;
 *this*.items = *new* Semaphore(0);
 *this*.spaces = *new* Semaphore(maxSize);
 *this*.mutex = *new* Semaphore(1);
 }

 *public void* put(*int* item) *throws* InterruptedException {
 spaces.acquire(); *// Wait for available space* mutex.acquire(); *// Ensure mutual exclusion* buffer.add(item);
 mutex.release();
 items.release(); *// Signal that an item is available* }

 *public int* take() *throws* InterruptedException {
 items.acquire(); *// Wait for an available item* mutex.acquire(); *// Ensure mutual exclusion
 int* item = buffer.poll();
 mutex.release();
 spaces.release(); *// Signal that space is available
 return* item;
 }
}

*public class* Consumer *implements Runnable* {
 *private final* SharedBuffer sharedBuffer;

 *public* Consumer(SharedBuffer sharedBuffer) {
 *this*.sharedBuffer = sharedBuffer;
 }

 @Override
 *public void* run() {
 *while* (*true*) {
 *try* {
 *int* item = sharedBuffer.take();
 System.*out*.println("Consuming " + item);
 Thread.*sleep*((*int*) (Math.*random*() \* 1000)); *// Simulate time taken to consume an item* } *catch* (InterruptedException e) {
 Thread.*currentThread*().interrupt();
 System.*out*.println("Consumer was interrupted");
 *break*; *// Exit the loop if interrupted* }
 }
 }
}

*public class* Producer *implements Runnable* {
 *private final* SharedBuffer sharedBuffer;

 *public* Producer(SharedBuffer sharedBuffer) {
 *this*.sharedBuffer = sharedBuffer;
 }

 @Override
 *public void* run() {
 *for* (*int* i = 0; i < 10; i++) {
 *try* {
 System.*out*.println("Producing " + i);
 sharedBuffer.put(i);
 Thread.*sleep*((*int*) (Math.*random*() \* 1000)); *// Simulate time taken to produce an item* } *catch* (InterruptedException e) {
 Thread.*currentThread*().interrupt();
 System.*out*.println("Producer was interrupted");
 }
 }
 }
}

*public class* ProducerConsumerExample {
 *public static void* main(String[] args) {
 SharedBuffer sharedBuffer = *new* SharedBuffer(5);

 Thread producerThread = *new* Thread(*new* Producer(sharedBuffer));
 Thread consumerThread = *new* Thread(*new* Consumer(sharedBuffer));

 producerThread.start();
 consumerThread.start();

 *try* {
 producerThread.join(); *// Wait for the producer to finish* consumerThread.interrupt(); *// Interrupt the consumer after the producer is done* consumerThread.join(); *// Wait for the consumer to finish* } *catch* (InterruptedException e) {
 Thread.*currentThread*().interrupt();
 System.*out*.println("Main thread was interrupted");
 }
 }
}

Producing 0

Consuming 0

Producing 1

Consuming 1

Producing 2

Consuming 2

Producing 3

Consuming 3

Producing 4

Producing 5

Consuming 4

Consuming 5

Producing 6

Consuming 6

Producing 7

Consuming 7

Producing 8

Producing 9

Consuming 8