<https://www.youtube.com/watch?v=x5SXXBXBtOc>

<https://www.youtube.com/watch?v=nO6bi7ESRCQ>

<https://www.youtube.com/watch?v=I0nHCNECynk>

Producer Consumer is also called as Bounded Buffer Problem

1. Producer Consumer using wait() and notify()
2. Producer Consumer using blockingQueue
3. Producer Consumer using semaphore

**Producer Consumer using wait() and notify()**

**package** test;

**import** java.util.Queue;

**import** java.util.Random;

**public** **class** Producer **implements** Runnable {

**public** Queue<Integer> sharedQueue;

**public** **final** **int** MAX\_SIZE = 5;

// Constructor

**public** Producer(Queue<Integer> sharedQueue) {

**super**();

**this**.sharedQueue = sharedQueue;

}

@Override

**public** **void** run() {

**while** (**true**) {

**synchronized** (sharedQueue) {

**while** (sharedQueue.size() == 5) {

**try** {

System.***out***.println("Producer is waiting for Consumer to consume objects");

sharedQueue.wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

Random random = **new** Random();

**int** data = random.nextInt(MAX\_SIZE);

sharedQueue.add(data);

System.***out***.println("Produced : " + data);

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

sharedQueue.notify();

}

}

}

}

**package** test;

**import** java.util.Queue;

**public** **class** Consumer **implements** Runnable {

**public** Queue<Integer> sharedQueue;

// Constructor

**public** Consumer(Queue<Integer> sharedQueue) {

**super**();

**this**.sharedQueue = sharedQueue;

}

@Override

**public** **void** run() {

**while** (**true**) {

**synchronized** (sharedQueue) {

**while** (sharedQueue.isEmpty()) {

**try** {

System.***out***.println("Consumer is waiting for Producer to produce objects");

sharedQueue.wait();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**int** data = sharedQueue.poll();

System.***out***.println("Consumed : " + data);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

sharedQueue.notify();

}

}

}

}

**package** test;

**import** java.util.LinkedList;

**import** java.util.Queue;

**public** **class** ProducerConsumerDemo {

**public** **static** **void** main(String[] args) {

Queue<Integer> sharedQueue = **new** LinkedList<>();

Producer producer = **new** Producer(sharedQueue);

Consumer consumer = **new** Consumer(sharedQueue);

Thread producerThread = **new** Thread(producer);

Thread consumerThread = **new** Thread(consumer);

producerThread.start();

consumerThread.start();

}

}

**Producer Consumer using blockingQueue**

**package** test;

**import** java.util.Random;

**import** java.util.concurrent.BlockingQueue;

**public** **class** Producer **implements** Runnable {

**public** BlockingQueue<Integer> sharedQueue;

**public** **final** **int** MAX\_SIZE = 5;

// Constructor

**public** Producer(BlockingQueue<Integer> sharedQueue) {

**super**();

**this**.sharedQueue = sharedQueue;

}

@Override

**public** **void** run() {

**while** (**true**) {

Random random = **new** Random();

**int** data = random.nextInt(MAX\_SIZE);

**try** {

sharedQueue.put(data);

System.***out***.println("Produced : " + data);

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**package** test;

**import** java.util.concurrent.BlockingQueue;

**public** **class** Consumer **implements** Runnable {

**public** BlockingQueue<Integer> sharedQueue;

**public** **final** **int** MAX\_SIZE = 5;

// Constructor

**public** Consumer(BlockingQueue<Integer> sharedQueue) {

**super**();

**this**.sharedQueue = sharedQueue;

}

@Override

**public** **void** run() {

**while** (**true**) {

**int** data;

**try** {

data = sharedQueue.take();

System.***out***.println("Consumed : " + data);

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**package** test;

**import** java.util.concurrent.ArrayBlockingQueue;

**import** java.util.concurrent.BlockingQueue;

**public** **class** ProducerConsumerDemo {

**public** **static** **void** main(String[] args) {

BlockingQueue<Integer> sharedQueue = **new** ArrayBlockingQueue<>(5);

Producer producer = **new** Producer(sharedQueue);

Consumer consumer = **new** Consumer(sharedQueue);

Thread producerThread = **new** Thread(producer);

Thread consumerThread = **new** Thread(consumer);

producerThread.start();

consumerThread.start();

}

}

**Producer Consumer using semaphore**

**package** test;

//Java implementation of a producer and consumer

//that use semaphores to control synchronization.

**import** java.util.concurrent.Semaphore;

**public** **class** MyQueue {

// an item

**private** **int** item;

// semaphoreConsumer initialized with 0 permits

// to ensure put() method executes first

**private** Semaphore semaphoreConsumer = **new** Semaphore(0);

**private** Semaphore semaphoreProducer = **new** Semaphore(1);

// to get an item from buffer

**public** **void** get() {

**try** {

// Before consumer can consume an item,

// it must acquire a permit from semaphoreConsumer

semaphoreConsumer.acquire();

} **catch** (InterruptedException e) {

System.***out***.println("InterruptedException caught");

}

// consumer consuming an item

System.***out***.println("Consumer consumed item : " + item);

// After consumer consumes the item,

// it releases semaphoreProducer to notify producer

semaphoreProducer.release();

}

// To put an item in buffer

**public** **void** put(**int** item) {

**try** {

// Before producer can produce an item,

// it must acquire a permit from semaphoreProducer

semaphoreProducer.acquire();

} **catch** (InterruptedException e) {

System.***out***.println("InterruptedException caught");

}

// producer producing an item

**this**.item = item;

System.***out***.println("Producer produced item : " + item);

// After producer produces the item,

// it releases semaphoreConsumer to notify consumer

semaphoreConsumer.release();

}

}

**package** test;

**import** java.util.Random;

**public** **class** Producer **implements** Runnable {

**private** MyQueue myQueue;

**public** Producer(MyQueue myQueue) {

**this**.myQueue = myQueue;

}

**public** **void** run() {

**while** (**true**) {

Random random = **new** Random();

**int** data = random.nextInt(100);

// producer put items

myQueue.put(data);

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**package** test;

**public** **class** Consumer **implements** Runnable {

**private** MyQueue myQueue;

**public** Consumer(MyQueue myQueue) {

**this**.myQueue = myQueue;

}

**public** **void** run() {

**while** (**true**) {

// consumer get items

myQueue.get();

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**package** test;

**public** **class** ProducerConsumerDemo {

**public** **static** **void** main(String[] args) {

// creating buffer queue

MyQueue myQueue = **new** MyQueue();

Producer producer = **new** Producer(myQueue);

Consumer consumer = **new** Consumer(myQueue);

Thread producerThread = **new** Thread(producer);

// starting producer thread

producerThread.start();

Thread consumerThread = **new** Thread(consumer);

// starting consumer thread

consumerThread.start();

}

}