Advanced Java: Multi-threading Part 7 - Producer-Consumer using BlockingInteger

https://www.youtube.com/watch?v=Vrt5LqpH2D0

**import** java.util.ArrayList;

**import** java.util.Random;

**import** java.util.Scanner;

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

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

**public** **class** apples{

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

**private** **volatile** **static** **boolean** *tof*=**true**;

**private** **static** **void** producer() **throws** InterruptedException{

**while**(*tof*){

*queue*.put(*random*.nextInt(100));

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

}

}

**private** **static** **void** consumer() **throws** InterruptedException{

**while**(*tof*){

Thread.*sleep*(100);

**if**(*random*.nextInt(10)==0){

Integer value = *queue*.take();

System.*out*.println("Value taken: "+value+"; Queue size: "+*queue*.size());

}

}**if**(!*tof*)*queue*.drainTo(**new** ArrayList<Integer>());

}

**private** **static** BlockingQueue<Integer> *queue* = **new** ArrayBlockingQueue<Integer>(10);

**private** **static** **void** stop(){

*tof*=**false**;

}

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

Scanner scanner = **new** Scanner(System.*in*);

System.*out*.println("Press Enter to terminate");

Thread t1 = **new** Thread(**new** Runnable(){

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

**try** {

*producer*();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

});

Thread t2 = **new** Thread(**new** Runnable(){

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

**try** {

*consumer*();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

});

t1.start();

t2.start();

scanner.nextLine();

*stop*();

t1.join();

t2.join();

System.*out*.println("Terminated");

}

}

//3 great things about this

//1.No synchronized keyword here

//2.In case there's nothing to take in queue, the take() method will patiently wait

//3. In case the queue is full, the put() method will also patiently wait.