ProducerConsumer Class:

```
package producerconsumerproblem;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class ProducerConsumerProblem {

    public static int buffSize;
    public static Queue<Integer> buffer = new LinkedList<>();

    public static void main(String[] args) {

        // Taking user input on what will be the size of the buffer
        System.out.println("Enter the required Buffer Size: ");
        Scanner sc = new Scanner(System.in);
        buffSize = sc.nextInt();

        /*
        ** Initializing 3 threads to be used.
        ** 1. stoppingThread -> A thread to stop the program execution on user's command
        ** 2. producerThread -> The thread to handle producer thread
        ** 3. consumerThread -> The thread to handle consumer thread
        */

        Thread stoppingThread = new Thread(new Stopping());
        Thread producerThread = new Thread(new Producer());
        Thread consumerThread = new Thread(new Consumer());

        // Start the threads
        producerThread.start();
        consumerThread.start();
        stoppingThread.start();
    }
}
```

Producer Class:

```
package producerconsumerproblem;

import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Producer implements Runnable {

    @Override
    public void run() {
        // Calling the produce method which will take care of the rest
        produce();
    }

    private void produce() {
        // Values to be entered in the buffer
        int val = 0;
        while (true) {
            try {
                // To control whether to wait, or keep on production
                while (ProducerConsumerProblem.buffer.size() == ProducerConsumerProblem.buffSize) {
                    synchronized (ProducerConsumerProblem.buffer) {
                        ProducerConsumerProblem.buffer.wait();
                    }
                }

                // Produce and store new values in the queue
                synchronized (ProducerConsumerProblem.buffer) {
                    System.out.println("Producer produced: " + val);
                    ProducerConsumerProblem.buffer.add(val++);
                    System.out.println("Current size of Buffer: " + ProducerConsumerProblem.buffer.size() + "\n");

                    ProducerConsumerProblem.buffer.notifyAll();
                }

                // Create a random number from 100 till 999. This will be sleep time denoting time needed before another produce entry
                Random rand = new Random();
                Thread.sleep(rand.nextInt(1000 - 100) + 100);
            } catch (InterruptedException ex) {
                Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

Consumer Class:

```
package producerconsumerproblem;

import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Consumer implements Runnable {

    @Override
    public void run() {
        // Calling the consumer method which will take care of the rest
        consume();
    }

    private void consume() {
        // Values to be removed in the buffer
        int val;
        while (true) {
            try {
                // To control whether to wait, or keep on consuming
                while (ProducerConsumerProblem.buffer.isEmpty()) {
                    synchronized (ProducerConsumerProblem.buffer) {
                        ProducerConsumerProblem.buffer.wait();
                    }
                }

                // Consume existing values from the queue
                synchronized (ProducerConsumerProblem.buffer) {

                    val = ProducerConsumerProblem.buffer.remove();
                    System.out.println("Consumer consumed: " + val + "\nCurrent size of Buffer: " + ProducerConsumerProblem.buffer.size() + "\n");

                    ProducerConsumerProblem.buffer.notifyAll();
                }

                // Create a random number from 100 till 999. This will be sleep time denoting time needed before another produce entry
                Random rand = new Random();
                Thread.sleep(rand.nextInt(1000 - 100) + 100);
            } catch (InterruptedException ex) {
                Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

Stopping Class:

```
package producerconsumerproblem;

import javax.swing.JDialog;
import javax.swing.JOptionPane;


public class Stopping implements Runnable{

    @Override
    public void run() {
        // The JOptionPane which will help user control when to stop the code
        final JDialog dialog = new JDialog();
        dialog.setAlwaysOnTop(true);
        JOptionPane.showMessageDialog(dialog, "Closing this will cause the program to stop");
        System.exit(0);
    }
}
```