Nikit Gokhe

Class – Comp D1

Roll No. 224024

Gr No. 21810522

ASSIGNMENT 4

AIM : Implement Producer/Consumer problem using java threads.

Theory: In computing, the producer-consumer problem (also known as the bounded- buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

- The producer’s job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

Problem

To make sure that the producer won’t try to add data into the buffer if it’s full and that the consumer won’t try to remove data from an empty buffer.

Solution

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

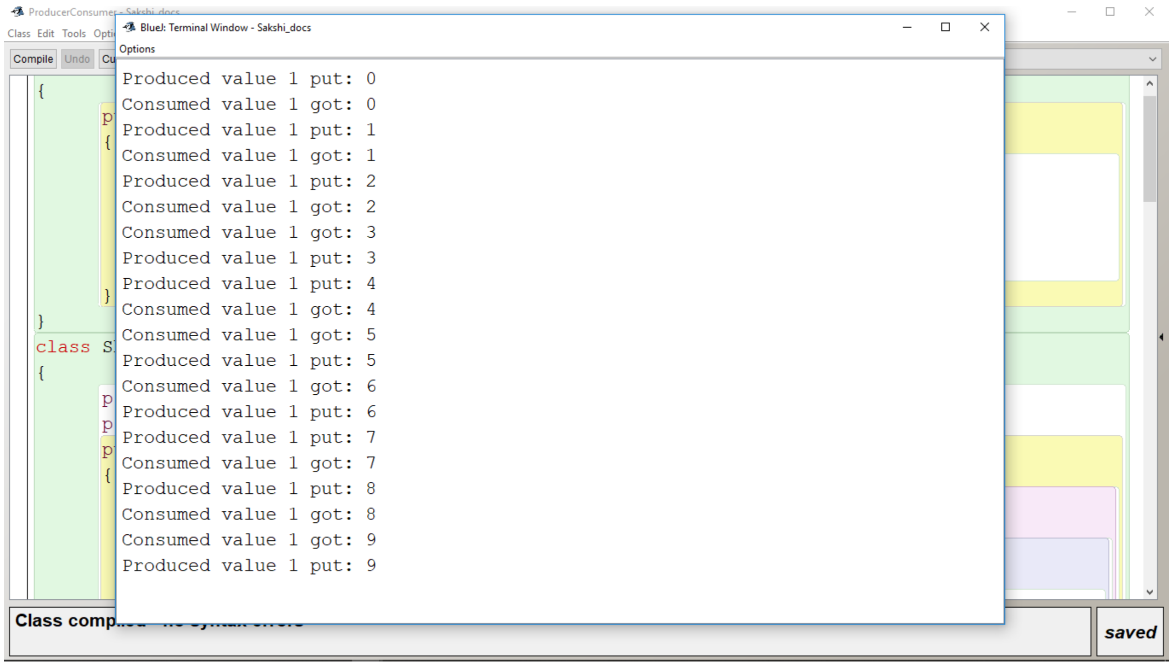
CODE :

```
public class ProducerConsumer
{
    public static void main(String[] args)
    {
        Shop c = new Shop();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start();
        c1.start();
    }
}
class Shop
{
    private int materials;
    private boolean available = false;
    public synchronized int get()
    {
        while (available == false)
        {
            try
            {
                wait();
            }
            catch (InterruptedException ie)
            {
            }
        }
```

```
            available = false;
            notifyAll();
            return materials;
        }
        public synchronized void put(int value)
        {
            while (available == true)
            {
                try
                {
                    wait();
                }
                catch (InterruptedException ie)
                {
                    ie.printStackTrace();
                }
            }
            materials = value;
            available = true;
            notifyAll();
        }
    }
    class Consumer extends Thread
    {
        private Shop Shop;
        private int number;
        public Consumer(Shop c, int number)
        {
            Shop = c;
            this.number = number;
        }
        public void run()
        {
            int value = 0;
            for (int i = 0; i < 10; i++)
            {
                value = Shop.get();
                System.out.println("Consumed value " + this.number+ " got: " + value);
            }
        }
    }
    class Producer extends Thread
    {
        private Shop Shop;
        private int number;

        public Producer(Shop c, int number)
        {
            Shop = c;
            this.number = number;
        }
        public void run()
        {
            for (int i = 0; i < 10; i++)
            {
                Shop.put(i);
                System.out.println("Produced value " + this.number+ " put: " + i);
                try
                {
                    sleep((int)(Math.random() * 100));
                }
                catch (InterruptedException ie)
                {
                    ie.printStackTrace();
                }
            }
    }
}
```

OUTPUT :



ProducerConsumer - Sakshi_docs
Class Edit Tools Opti
Compile Undo Cu
BlueJ: Terminal Window - Sakshi_docs
Options
Class comp
saved

```
Produced value 1 put: 0
Consumed value 1 got: 0
Produced value 1 put: 1
Consumed value 1 got: 1
Produced value 1 put: 2
Consumed value 1 got: 2
Consumed value 1 got: 3
Produced value 1 put: 3
Produced value 1 put: 4
Consumed value 1 got: 4
Consumed value 1 got: 5
Produced value 1 put: 5
Consumed value 1 got: 6
Produced value 1 put: 6
Produced value 1 put: 7
Consumed value 1 got: 7
Produced value 1 put: 8
Consumed value 1 got: 8
Consumed value 1 got: 9
Produced value 1 put: 9
```