

## Practical 8

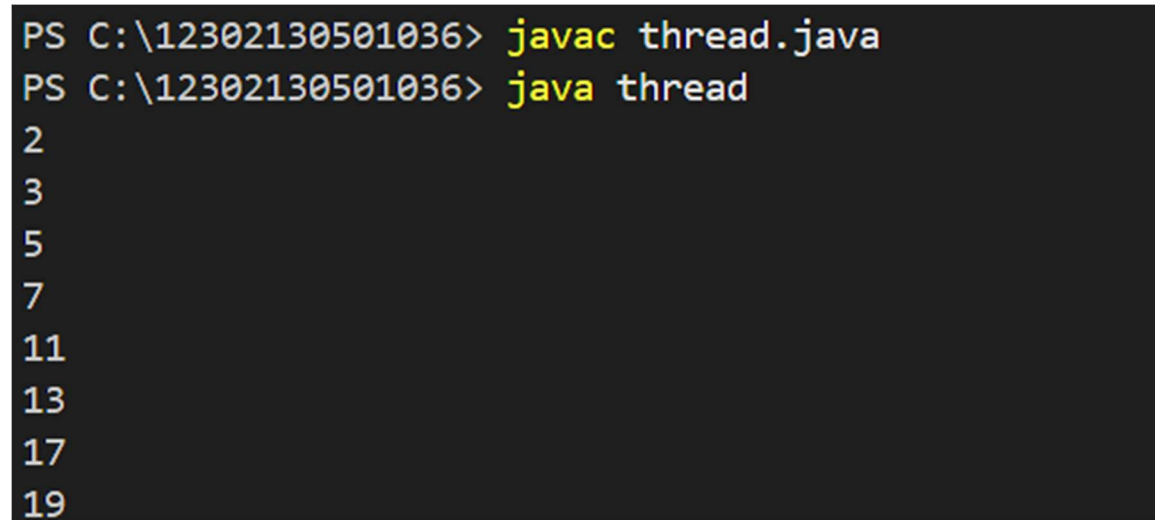
1. Write a program to find prime number in given range using both method of multithreading. Also run the same program using executor framework

**Solution:**

```
class MyThread extends Thread {  
    int n;  
  
    MyThread(int n) {  
        this.n = n;  
    }  
  
    @Override  
    public void run() {  
        for (int i = 2; i < n; i++) {  
            int counter = 0;  
            for (int num = 1; num <= i; num++) {  
                if (i % num == 0) {  
                    counter++;  
                }  
            }  
            if (counter == 2) {  
                System.out.println(i);  
            }  
        }  
    }  
}  
  
public class thread {
```

```
public static void main(String[] args) {  
    MyThread m = new MyThread(100);  
    m.start();  
}  
}
```

Output:



```
PS C:\12302130501036> javac thread.java  
PS C:\12302130501036> java thread  
2  
3  
5  
7  
11  
13  
17  
19
```

2. Assume one class Queue that defines queue of fix size says 15.

- Assume one class producer which implements Runnable, having priority NORM\_PRIORITY +1
- One more class consumer implements Runnable, having priority NORM\_PRIORITY-1
- Class TestThread is having main method with maximum priority, which creates 1 thread for producer and 2 threads for consumer.
- Producer produces number of elements and put on the queue. when queue becomes full it notifies other threads.  
Consumer consumes number of elements and notifies other thread when queue become empty.

Solution:

```
class Queue {  
    private int[] queueArray;  
    public int front;
```

```
public int rear;

public Queue() {
    queueArray = new int[15];
    front = -1;
    rear = -1;
}

public synchronized int enqueue(int item) {
    if (isEmpty()) {
        front = 0;
        rear = 0;
        queueArray[rear] = item;
    } else {
        rear = (rear + 1) % 15;
        if (rear == front) {
            return -1;
        } else {
            queueArray[rear] = item;
        }
    }
    return 1;
}

public synchronized int dequeue() {
    int item = -1;
    if (!isEmpty()) {
        item = queueArray[front];

        if (front == rear) {
```

```
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % 15;
    }
}
return item;
}

public boolean isEmpty() {
    return front == -1 && rear == -1;
}
}

class Producer implements Runnable {
    private static int counter = 1;
    private static Queue queue;

    Producer(Queue q) {
        queue = q;
    }

    public void run() {
        while (true) {
            if (queue.enqueue(counter) == -1) {
                break;
            } else {
                System.out.println("Produced: " + counter);
                counter++;
            }
        }
    }
}
```

```
        System.out.println("Final Front Index: " + queue.front);
    }
}

class Consumer implements Runnable {
    private static Queue queue;

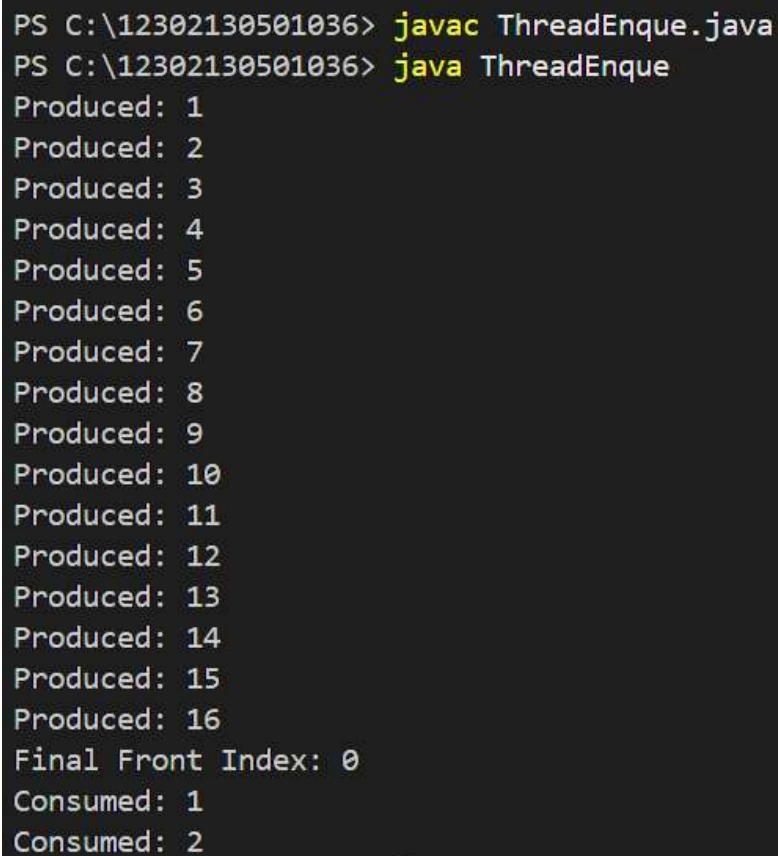
    Consumer(Queue q) {
        queue = q;
    }

    public void run() {
        while (true) {
            int item = queue.dequeue();
            if (item == -1) {
                break;
            } else {
                System.out.println("Consumed: " + item);
            }
        }
    }
}

public class ThreadEnque {
    public static void main(String[] args) {
        Queue q = new Queue();
        Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
        Thread producerThread = new Thread(new Producer(q));
        producerThread.setPriority(Thread.NORM_PRIORITY + 1);
        Thread consumerThread1 = new Thread(new Consumer(q));
    }
}
```

```
        consumerThread1.setPriority(Thread.NORM_PRIORITY - 1);  
        Thread consumerThread2 = new Thread(new Consumer(q));  
        consumerThread2.setPriority(Thread.NORM_PRIORITY - 1);  
        producerThread.start();  
        consumerThread1.start();  
        consumerThread2.start();  
    }  
}
```

Output:



```
PS C:\12302130501036> javac ThreadEnque.java  
PS C:\12302130501036> java ThreadEnque  
Produced: 1  
Produced: 2  
Produced: 3  
Produced: 4  
Produced: 5  
Produced: 6  
Produced: 7  
Produced: 8  
Produced: 9  
Produced: 10  
Produced: 11  
Produced: 12  
Produced: 13  
Produced: 14  
Produced: 15  
Produced: 16  
Final Front Index: 0  
Consumed: 1  
Consumed: 2
```