CO6

1. Define 2 classes; one for generating Fibonacci numbers and other for displaying even numbers in a given range. Implement using threads.

```
import java.util.ArrayList;
import java.util.List;

class FibonacciGenerator extends Thread {
    private int n;
    private List<Integer> fibSequence;

    public FibonacciGenerator(int n) {
        this.n = n;
        this.fibSequence = new ArrayList<>();
    }

    @Override
    public void run() {
        int a = 0, b = 1;
        for (int i = 0; i < n; i++) {
            fibSequence.add(a);
            int temp = a + b;
            a = b;
            b = temp;
        }
        System.out.println("Fibonacci Sequence (" + n + " terms): " +
fibSequence);
    }
}

class EvenNumberPrinter extends Thread {
    private int start;
    private int end;

    public EvenNumberPrinter(int start, int end) {
```

```
        this.start = start;
        this.end = end;
    }

    @Override
    public void run() {
        List<Integer> evenNumbers = new ArrayList<>();
        for (int i = start; i <= end; i++) {
            if (i % 2 == 0) {
                evenNumbers.add(i);
            }
        }
        System.out.println("Even Numbers in Range (" + start + " to " + end +
"): " + evenNumbers);
    }
}

public class Fiboeven {
    public static void main(String[] args) {
        int n = 10;
        int startNum = 20;
        int endNum = 40;

        FibonacciGenerator fibThread = new FibonacciGenerator(n);
        EvenNumberPrinter evenThread = new
EvenNumberPrinter(startNum, endNum);

        fibThread.start();
        evenThread.start();

        try {
            fibThread.join();
            evenThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
        System.out.println("Main Thread Exiting");
    }
}
```

OUTPUT

```
Fibonacci Sequence (10 terms): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Even Numbers in Range (20 to 40): [20, 22, 24, 26, 28, 30, 32, 34, 36, 38,
40]
Main Thread Exiting
```

2. Producer/Consumer using ITC

```
import java.util.LinkedList;
public class ProdCons {
public static void main(String[] args)
throws InterruptedException
{
// Object of a class that has both produce()
// and consume() methods
final PC pc = new PC();
// Create producer thread
Thread t1 = new Thread(new Runnable() {

@Override
public void run()
{
try {
pc.produce();
}
catch (InterruptedException e) {
e.printStackTrace();
}
}
});
// Create consumer thread
```

```
Thread t2 = new Thread(new Runnable() {
@Override
public void run()
{
try {
pc.consume();
}
catch (InterruptedException e) {
e.printStackTrace();
}
}
});
// Start both threads
t1.start();
t2.start();
// t1 finishes before t2
t1.join();
t2.join();
}
// This class has a list, producer (adds items to list
// and consumer (removes items).
public static class PC {
// Create a list shared by producer and consumer
// Size of list is 2.
LinkedList <Integer> list = new LinkedList <>();
int capacity = 2;
// Function called by producer thread
public void produce() throws InterruptedException
{
int value = 0;
int counter=1;
while (counter <= 11) {
synchronized (this)
{
// producer thread waits while list
// is full
if (list.size() == capacity)
```

```
   wait();
System.out.println("Producer produced- "+ value);
// to insert the jobs in the list

list.add(value++);
// notifies the consumer thread that
// now it can start consuming
notify();
// makes the working of program easier
// to understand
Thread.sleep(2000);
}
++counter;
}
}
// Function called by consumer thread
public void consume() throws InterruptedException
{
int counter=0;
while (counter <= 11) {
synchronized (this)
{
// consumer thread waits while list
// is empty
if (list.size() == 0)
wait();
// to retrieve the first job in the list
int val = list.removeFirst();
System.out.println("Consumer consumed- "+ val);
// Wake up producer thread
notify();
// and sleep
Thread.sleep(2000);
}
}
}
}
```

```
}
```

OUTPUT

```
Producer produced- 0
Producer produced- 1
Consumer consumed- 0
Consumer consumed- 1
Producer produced- 2
Producer produced- 3
Consumer consumed- 2
Consumer consumed- 3
Producer produced- 4 ……
```

3. Copying file using stream and character class

```
import java.io.*;
import java.util.*;
public class co63 {

        public static void copyData(File file1, File file2) throws Exception
        {
          FileInputStream inputStream = new FileInputStream(file1);
          FileOutputStream outputStream = new
FileOutputStream(file2);

          try {

            int i;
            while ((i = inputStream.read()) != -1) {

              outputStream.write(i);
            }
          }

          catch(Exception e) {
            System.out.println("Error Found: "+e.getMessage());
          }
```

```
            finally {
               if (inputStream != null) {
                  inputStream.close();
               }
               if (outputStream != null) {
                  outputStream.close();
               }
            }
            System.out.println("File Copied");
         }
         public static void main(String[] args) throws Exception
         {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the name of the file from where the
data would be copied :");
            String file1 = sc.nextLine();
            File a = new File("C:\\Users\\gauri\\Documents\\"+file1);
            System.out.println("Enter the name of the file from where the
data would be written :");
            String file2 = sc.nextLine();
            File b = new File("C:\\Users\\gauri\\Documents\\"+file2);
            sc.close();
            copyData(a, b);
         }
      }
```

OUTPUT

```
Enter the name of the file from where the data would be copied :
myfile.txt
Enter the name of the file from where the data would be written :
final.txt
File Copied
```