OS LAB: SEMAPHORES

Name: Disha Khater
Roll no. 64
12010067
CS-A

1. Producer-Consumer Problem

```
package OS_LABS;

import java.util.LinkedList;

public class Threads {
    public static void main(String[] args)
            throws InterruptedException
    {
        final PC pc = new PC();
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                try {
                    pc.produce();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                try {
                    pc.consume();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        t1.start();
        t2.start();

        t1.join();
        t2.join();
    }

    public static class PC {

        LinkedList<Integer> list = new LinkedList<>();
        int capacity = 2;

        public void produce() throws InterruptedException
```

```
        {
            int value = 0;
            while (true) {
                synchronized (this)
                {

                    while (list.size() == capacity)
                        wait();

                    System.out.println("Producer produced-"
                            + value);

                    list.add(value++);

                    notify();

                    Thread.sleep(1000);
                }
            }
        }

        public void consume() throws InterruptedException
        {
            while (true) {
                synchronized (this)
                {
                    while (list.size() == 0)
                        wait();

                    int val = list.removeFirst();

                    System.out.println("Consumer consumed-"
                            + val);

                    notify();

                    Thread.sleep(1000);
                }
            }
        }
    }
}
```

OUTPUT:

```
Producer produced-0
Producer produced-1
Consumer consumed-0
Producer produced-2
Consumer consumed-1
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Consumer consumed-4
Producer produced-5
Consumer consumed-5
Producer produced-6
Consumer consumed-6
Producer produced-7
Producer produced-8
Consumer consumed-7
Consumer consumed-8
Producer produced-9
Producer produced-10
Consumer consumed-9
Consumer consumed-10
```

2. Reader writer

```
package OS_LABS;

class controller
{
    int areader=0,awriter=0,wwriter=0,wreader=0;
    public int allowreader()
    {
        int res=0;
        if(wreader==0&&awriter==0)
        {
            res=1;
        }
        return res;
    }
    public int allowwriter()
    {
        int res=0;
        if(areader==0&&awriter==0)
        {
            res=1;
        }
        return res;
    }

    synchronized void beforeread()
    {
        wreader++;
        while(allowreader()!=0)
        {
            try
```

```
            {
                wait();
            }
            catch(InterruptedException e)
            {
            }
            wreader--;
            areader++;
        }
    }

    synchronized void beforewrite()
    {
        wwriter++;
        while(allowwriter()!=0)
        {
            try
            {
                wait();
            }
            catch(InterruptedException e)
            {
            }
            wwriter--;
            awriter++;
        }
    }

    synchronized void afterread()
    {
        areader--;
        notifyAll();
    }

    synchronized void afterwrite()
    {
        awriter--;
        notifyAll();
    }

}


class rew
{
    static controller ctl;

    public static void main(String[] args)
    {
        ctl=new controller();
        new reader1(ctl).start();
        new reader2(ctl).start();
        new writer1(ctl).start();
        new writer2(ctl).start();
    }
}

class reader1 extends Thread
{
    controller ctl;
    public reader1(controller c)
```

```
    {
        ctl=c;
    }
    public void run()
    {
        while(true)
        {
            ctl.beforeread();
            System.out.println("Reader1 Reading");
            System.out.println("Done Reading");
            ctl.afterread();
            System.out.println("After Read");
        }
    }
}
class reader2 extends Thread
{
    controller ctl;
    public reader2(controller c)
    {
        ctl=c;
    }
    public void run()
    {
        while(true)
        {
            ctl.beforeread();
            System.out.println("Reader2 Reading");
            System.out.println("Done Reading");
            ctl.afterread();
            System.out.println("After Read");
        }
    }
}

class writer1 extends Thread
{
    controller ctl;
    public writer1(controller c)
    {
        ctl=c;
    }
    public void run()
    {
        while(true)
        {
            ctl.beforewrite();
            System.out.println("Writer1 Writing");
            System.out.println("Done Writing");
            ctl.afterread();
            System.out.println("After Write");
        }
    }
}

class writer2 extends Thread
{
    controller ctl;
    public writer2(controller c)
    {
        ctl=c;
```

```
    }
    public void run()
    {
        while(true)
        {
            ctl.beforewrite();
            System.out.println("Writer2 Writing");
            System.out.println("Done Writing");
            ctl.afterread();
            System.out.println("After Write");
        }
    }
}
```

OUTPUT:

```
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
After Write
Writer2 Writing
Done Writing
```

3. Dinning Philosopher

```
package OS_LABS;

import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadLocalRandom;
public class DiningPhilosophersProblem
{
    static int philosopher = 5;
    static Philosopher philosophers[] = new Philosopher[philosopher];
    static Chopstick chopsticks[] = new Chopstick[philosopher];
    static class Chopstick
    {
        public Semaphore mutex = new Semaphore(1);
        void grab()
        {
            try
            {
                mutex.acquire();
```

```
            }
            catch (Exception e)
            {
                e.printStackTrace(System.out);
            }
        }
        //release the chopstick
        void release()
        {
            mutex.release();
        }
        //checks if the chopstick is free or not
        boolean isFree()
        {
            return mutex.availablePermits() > 0;
        }
    }  //end of Chopstick class
    static class Philosopher extends Thread
    {
        public int number;
        public Chopstick leftchopstick;
        public Chopstick rightchopstick;
        Philosopher(int num, Chopstick left, Chopstick right)
        {
            number = num;
            leftchopstick = left;
            rightchopstick = right;
        }
        public void run()
        {
            while (true)
            {
                leftchopstick.grab();
                System.out.println("Philosopher " + (number+1) + " grabs
left chopstick.");
                rightchopstick.grab();
                System.out.println("Philosopher " + (number+1) + " grabs
right chopstick.");
                eat();
                leftchopstick.release();
                System.out.println("Philosopher " + (number+1) + " releases
left chopstick.");
                rightchopstick.release();
                System.out.println("Philosopher " + (number+1) + " releases
right chopstick.");
            }
        }
        void eat()
        {
            try
            {
                int sleepTime = ThreadLocalRandom.current().nextInt(0,
1000);
                System.out.println("Philosopher " + (number+1) + " eats for
" + sleepTime +"ms"); //sleeps the thread for a specified time
                Thread.sleep(sleepTime);
            }
            catch (Exception e)
            {
                e.printStackTrace(System.out);
            }
```

```
        }
    }
    public static void main(String args[])
    {
        for (int i = 0; i < philosopher; i++)
        {
            chopsticks[i] = new Chopstick();
        }
        for (int i = 0; i < philosopher; i++)
        {
            philosophers[i] = new Philosopher(i, chopsticks[i],
chopsticks[(i + 1) % philosopher]);
            philosophers[i].start();
        }
        while (true)
        {
            try
            {
                Thread.sleep(1000);
                boolean deadlock = true;
                for (Chopstick cs : chopsticks)
                {
                    if (cs.isFree())
                    {
                        deadlock = false;
                        break;
                    }
                }
                if (deadlock)
                {
                    Thread.sleep(1000);
                    System.out.println("Everyone Eats");
                    break;
                }
            }
            catch (Exception e)
            {
                e.printStackTrace(System.out);
            }
        }
        System.out.println("Exit The Program!");
        System.exit(0);
    }
}
```

OUTPUT:

```
Philosopher 3 grabs left chopstick.
Philosopher 4 grabs left chopstick.
Philosopher 5 grabs left chopstick.
Philosopher 2 grabs left chopstick.
Philosopher 1 grabs left chopstick.
Everyone Eats
Exit The Program!

Process finished with exit code 0
```