LAB-10

0 Demonstrate interprocess communication and deadlock

```
class Q {
    int n;
    boolean valueSet = false;
    synchronized int get() {
        while(!valueSet)
        try {
            System.out.println("\n Consumer Waiting");
            wait();
        }
        catch(InterruptedException e) {
            System.out.println("Interrupted Exception");
        }
        System.out.println("Got: " + n);
        valueSet = false;
        System.out.println("Intimate Producer\n");
        notify();
        return n;
    }
    synchronized void put(int n) {
        while(valueSet)
            try {
                System.out.println("Producer Waiting");
                wait();
            }
            catch (InterruptedException e)
            {
                System.out.println("Interrupted Exception
                caught");
            }
```

```
        this.n = n;
        valueSet = true;
        System.out.println("Put: " + n);
        System.out.println("Intimate Consumer");
        notify();
    }
}

class Producer implements Runnable {
    Q q;
    Producer(Q q) {
        this.q = q;
        new Thread(this, "Producer");
        start();
    }
    public void run() {
        int i = 0;
        while(i<15) {
            q.put(i++);
        }
    }
}

class Consumer implements Runnable {
    Q q;
    Consumer(Q q) {
        this.q = q;
        new Thread(this, "Consumer");
        start();
    }
    public void run() {
        int i = 0;
        while(i<15) {
            int r = q.get();
```

```
System.out.println("consumed : " + n);
i++;
}
}
}
class PCFixed {
public static void main(String args[])
{
Q q = new Q();
new Producer(q);
new Consumer(q);
}
}
```

Output:

Put : 0
Intimate consumer
Producer Waiting
Get : 0
Intimate Producer
consumed : 0
Consumer Waiting
Put : 1
Intimate Consumer
Producer Waiting
Get : 1
Intimate Producer
consumed : 1
Put : 2
Intimate consumer
Producer Waiting
Get : 2
Intimate Consumer
Producer Waiting
Put 3:
Intimate Waiting
Get : 3
Intimate Consumer
Consumed : 3
Put : 4

Deadlocking :→

```
class A {
    synchronized void foo(B b) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered A.foo");
        try {
            Thread.sleep(1000);
        }
        catch(Exception e) {
            System.out.println("A interrupted");
        }
        System.out.println(name + " trying to call B.last()");
        b.last();
    }
    void last() {
        System.out.println("Inside A.last");
    }
}
class B {
    synchronized void bar(A a) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered B.bar");
        try {
            Thread.sleep(1000);
        }
        catch(Exception e) {
            System.out.println("B interrupted");
        }
```

```
        System.out.println(name + " trying to call A.last()");
        a.last();
    }
    void last() {
        System.out.println("Inside A.last");
    }
}
class Deadlock implements Runnable {
    A a = new A();
    B b = new B();
    Deadlock() {
        Thread.currentThread().setName("MainThread");
        Thread t = new Thread(this, "RacingThread");
        t.start();
        a.foo(b);
        System.out.println("Back in main thread");
    }
public void run() {
    b.bar(a);
    System.out.println("Back in other thread");
}
public static void main(String args[]) {
    new Deadlock();
}
}
```

Output :-

Main Thread entered A.foo

Racing Thread entered B.bar

Main thread trying to call B.last()

Inside B.last

Back in main thread

Racing Thread trying to call A.last()

Inside A.last

Back in other thread

13.02.24