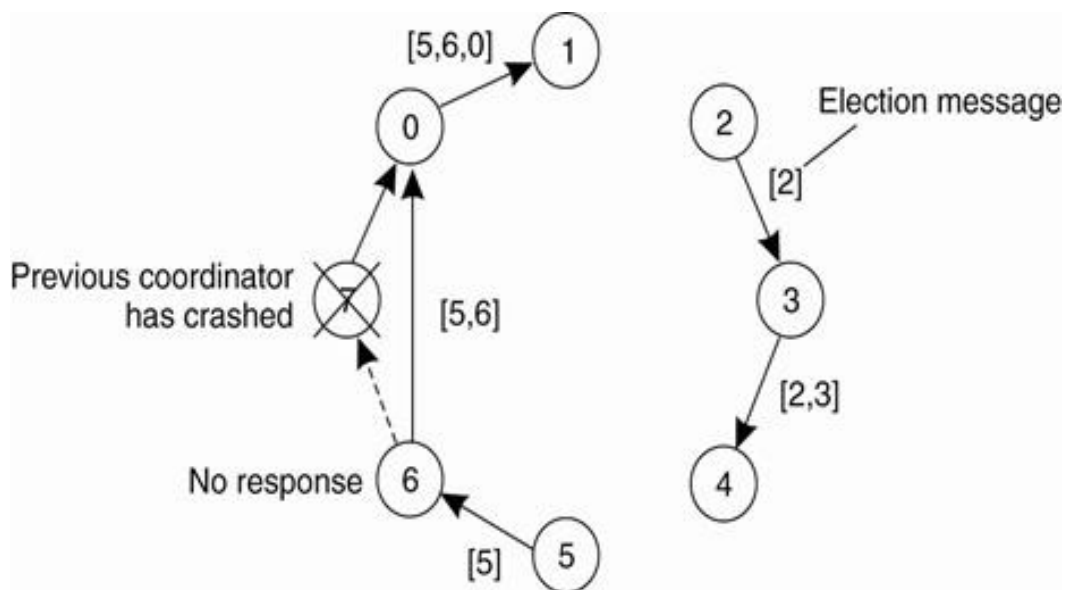


Aim: Program to implement Ring Election Algorithm.

Another election algorithm is based on the use of a ring, but without a token. We assume that the processes are physically or logically ordered, so that each process knows who its successor is. When any process notices that the coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down, the sender skips over the successor and goes to the next member along the ring, or the one after that, until a running process is located. At each step, the sender adds its own process number to the list in the message. Eventually, the message gets back to the process that started it all. That process recognizes this event when it receives an incoming message containing its own process number. At that point, the message type is changed to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is (the list member with the highest number) and who the members of the new ring are. When this message has circulated once, it is removed and everyone goes back to work.



Code:

```
import java.util.Scanner;

class Process {
    public int id;
    public boolean active;

    public Process(int id) {
        this.id = id;
        active = true;
    }
}

public class Ring {
    int noOfProcesses;
    Process[] processes;
    Scanner sc;

    public Ring() {
        sc = new Scanner(System.in);
    }

    public void initialiseRing() {
        System.out.println("Enter no of processes");

        noOfProcesses = sc.nextInt();
        processes = new Process[noOfProcesses];
        for (int i = 0; i < processes.length; i++) {
            processes[i] = new Process(i);
        }
    }

    public int getMax() {
        int maxId = -99;
        int maxIdIndex = 0;
        for (int i = 0; i < processes.length; i++) {
            if (processes[i].active && processes[i].id > maxId) {
                maxId = processes[i].id;
                maxIdIndex = i;
            }
        }
        return maxIdIndex;
    }

    public void performElection() {

        System.out.println("Process no " + processes[getMax()].id + " fails");
    }
}
```

```

processes[getMax()].active = false;
System.out.println("Election Initiated by");
int initiatorProcesss = sc.nextInt();

int prev = initiatorProcesss;
int next = prev + 1;

while (true) {
    if (processes[next].active) {
        System.out.println("Process " + processes[prev].id + " pass
Election(" + processes[prev].id + ") to "
        + processes[next].id);
        prev = next;
    }

    next = (next + 1) % noOfProcesses;
    if (next == initiatorProcesss) {
        break;
    }
}

System.out.println("Process " + processes[getMax()].id + " becomes
coordinator");
int coordinator = processes[getMax()].id;

prev = coordinator;
next = (prev + 1) % noOfProcesses;

while (true) {

    if (processes[next].active) {
        System.out.println("Process " + processes[prev].id + " pass
Coordinator(" + coordinator
        + ") message to process " + processes[next].id);
        prev = next;
    }
    next = (next + 1) % noOfProcesses;
    if (next == coordinator) {
        System.out.println("End Of Election ");
        break;
    }
}

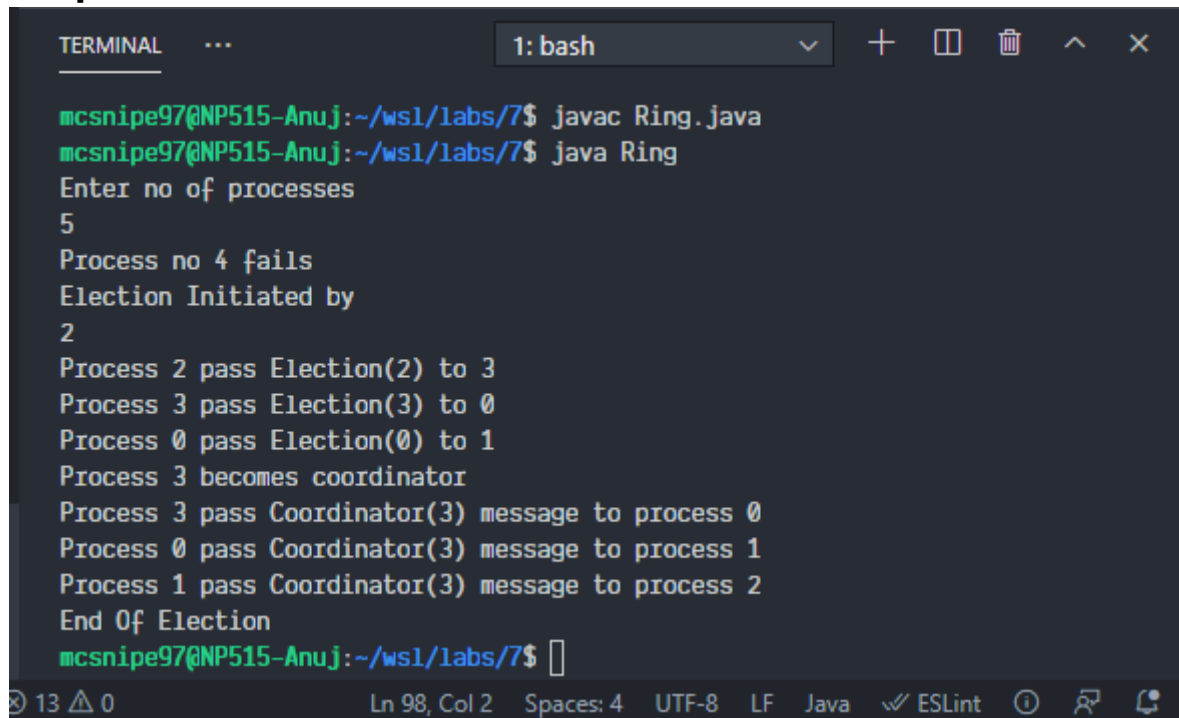
}

public static void main(String arg[]) {
    Ring r = new Ring();
    r.initialiseRing();
}

```

```
        r.performElection();
    }
}
```

Output:



```
TERMINAL  ...  1: bash  +  []  🗑  ^  x

mcsnipe97@NP515-Anuj:~/wsl/labs/7$ javac Ring.java
mcsnipe97@NP515-Anuj:~/wsl/labs/7$ java Ring
Enter no of processes
5
Process no 4 fails
Election Initiated by
2
Process 2 pass Election(2) to 3
Process 3 pass Election(3) to 0
Process 0 pass Election(0) to 1
Process 3 becomes coordinator
Process 3 pass Coordinator(3) message to process 0
Process 0 pass Coordinator(3) message to process 1
Process 1 pass Coordinator(3) message to process 2
End Of Election
mcsnipe97@NP515-Anuj:~/wsl/labs/7$
```

Conclusion:

We successfully implemented Ring Election Algorithm.