```
import java.util.Scanner;

class BullyAlgorithm {
    static int numProcesses;
    static boolean[] alive;
    static int coordinator;

    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of processes: ");
        numProcesses = sc.nextInt();
        alive = new boolean[numProcesses];

        for (int i = 0; i < numProcesses; i++) {
            alive[i] = true;
        }

        System.out.print("Enter the initial coordinator (0 to " + (numProcesses - 1) + "): ");
        coordinator = sc.nextInt();

        if (coordinator < 0 || coordinator >= numProcesses) {
            System.out.println("Invalid coordinator ID.");
            return;
        }

        System.out.println("Initial Coordinator: Process " + coordinator);

        System.out.print("Enter the process to crash (0 to " + (numProcesses - 1) + "): ");
        int crash = sc.nextInt();
```

```
    if (crash < 0 || crash >= numProcesses) {
        System.out.println("Invalid process ID to crash.");
        return;
    }

    alive[crash] = false;
    System.out.println("Process " + crash + " has crashed.");

    if (crash == coordinator) {
        System.out.println("Coordinator has crashed. Starting election...");
        System.out.print("Enter the process to start the election: ");
        int initiator = sc.nextInt();

        if (initiator < 0 || initiator >= numProcesses || !alive[initiator]) {
            System.out.println("Invalid or crashed initiator process.");
        } else {
            elect(initiator);
        }
    } else {
        System.out.println("Coordinator is still alive. No election needed.");
    }

    sc.close();
}

static void elect(int initiator) throws InterruptedException {
    System.out.println("\nProcess " + initiator + " is initiating an election...");
    boolean higherExists = false;

    for (int i = initiator + 1; i < numProcesses; i++) {
        System.out.print("Process " + initiator + " -> Process " + i + " (ELECTION)");
```

```
            if (alive[i]) {
                System.out.println();
                higherExists = true;
            } else {
                System.out.println(" - No response (Process " + i + " is crashed)");
            }
        }

        if (!higherExists) {
            declareWinner(initiator);
        } else {
            Thread.sleep(500);
            for (int i = initiator + 1; i < numProcesses; i++) {
                if (alive[i]) {
                    System.out.println("Process " + i + " -> Process " + initiator + " (OK)");
                }
            }
            for (int i = initiator + 1; i < numProcesses; i++) {
                if (alive[i]) {
                    elect(i);
                    return;
                }
            }
        }
    }

    static void declareWinner(int winner) throws InterruptedException {
        coordinator = winner;
        System.out.println("\nProcess " + winner + " wins the election and becomes the new
coordinator.");
        for (int i = 0; i < numProcesses; i++) {
```

```
            if (i == winner)
                continue;
            if (alive[i]) {
                System.out.println("Process " + winner + " -> Process " + i + " (COORDINATOR)");
                Thread.sleep(500);
            } else {
                System.out.println("Process " + i + " is crashed and did not receive COORDINATOR
message.");
            }
        }
    }
}
```

2)RING

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

class RingElection {
    private int numProcesses;
    private int coordinator;
    public boolean[] activeProcesses;

    public RingElection(int numProcesses, int coordinator) {
        this.numProcesses = numProcesses;
        this.activeProcesses = new boolean[numProcesses];

        // Initialize all processes as active
        for (int i = 0; i < numProcesses; i++) {
            activeProcesses[i] = true;
        }
```

```
    this.coordinator = coordinator;
    System.out.println("Initial Coordinator: Process " + coordinator);
}

public void simulateCrash(int processId) {
    if (processId >= 0 && processId < numProcesses && activeProcesses[processId]) {
        activeProcesses[processId] = false;
        System.out.println("Process " + processId + " has crashed!");
    } else {
        System.out.println("Invalid or already crashed process.");
    }
}

public void startElection(int initiator) {
    if (!activeProcesses[initiator]) {
        System.out.println("Process " + initiator + " is crashed and cannot start the election.");
        return;
    }

    System.out.println("\nProcess " + initiator + " is initiating an election...");
    List<Integer> electionPath = new ArrayList<>();
    electionPath.add(initiator);
    System.out.println("Election path: " + electionPath);

    int maxId = initiator;
    int current = (initiator + 1) % numProcesses;

    while (current != initiator) {
        if (activeProcesses[current]) {
            System.out.println("Process " + maxId + " -> Process " + current + " (ELECTION)");
            electionPath.add(current);
```

```
            System.out.println("Election path: " + electionPath);
            if (current > maxId) {
                maxId = current;
            }
        } else {
            System.out.println("Process " + current + " is skipped (CRASHED).");
        }
        current = (current + 1) % numProcesses;
    }

    coordinator = maxId;
    System.out.println("\nProcess " + coordinator + " wins the election and becomes the new coordinator.");
    announceNewCoordinator();
  }

  private void announceNewCoordinator() {
    int current = (coordinator + 1) % numProcesses;
    while (current != coordinator) {
        if (activeProcesses[current]) {
            System.out.println("Process " + coordinator + " -> Process " + current + " (ELECTED)");
        }
        current = (current + 1) % numProcesses;
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the number of processes: ");
    int numProcesses = scanner.nextInt();
```

```
System.out.print("Enter the initial coordinator (0 to " + (numProcesses - 1) + "): ");
int coordinator = scanner.nextInt();

while (coordinator < 0 || coordinator >= numProcesses) {
   System.out.print("Invalid coordinator. Enter again: ");
   coordinator = scanner.nextInt();
}

RingElection ring = new RingElection(numProcesses, coordinator);

System.out.print("Enter a process to crash: ");
int crash = scanner.nextInt();
ring.simulateCrash(crash);

// Only check and stop if the initial coordinator is crashed
if (!ring.activeProcesses[coordinator]) {
   System.out.println(
         "The initial coordinator (Process " + coordinator + ") has crashed. Election cannot start.");
   return; // Stop further execution if the initial coordinator has crashed
}

// No message if other processes have crashed
System.out.print("\nEnter the process to start the election: ");
int initiator = scanner.nextInt();

while (initiator < 0 || initiator >= numProcesses || !ring.activeProcesses[initiator]) {
   System.out.print("Invalid or crashed process. Enter a valid process to start the election: ");
   initiator = scanner.nextInt();
}
```

```
        ring.startElection(initiator);
    }
}
```