# DEADLOCK SIMULATION

| S.No | Team Name | Roll number |
|---|---|---|
| 1 | Ehtesham Ali Haidar | CB.EN.U4CSE22511 |
| 2 | Kuruboor Venkatesha Deepak | CB.EN.U4CSE22521 |
| 3 | Narravula mukesh | CB.EN.U4CSE22531 |

This is the simulation for deadlock for a real time general scenario :

Java code :

```
import java.util.Scanner;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class DeadlockSimulation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter number of resources: ");
        int resources = scanner.nextInt();
        System.out.print("Enter number of processes: ");
        int processes = scanner.nextInt();
        scanner.close();
        // Initialize locks for resources
        Lock[] locks = new ReentrantLock[resources];
        for (int i = 0; i < resources; i++) {
            locks[i] = new ReentrantLock();
```

```
        }
        // Crate and start process threads
        Thread[] threads = new Thread[processes];
        for (int i = 0; i < processes; i++) {
            threads[i] = new Thread(new Deadlock(i, locks, resources));
            threads[i].start();
        }
    }
}
class Deadlock implements Runnable {
    private final int index;
    private final Lock[] locks;
    private final int resources;

    public Deadlock(int index, Lock[] locks, int resources) {
        this.index = index;
        this.locks = locks;
        this.resources = resources;
    }
    @Override
    public void run() {
        int firstLock = index % resources;
        int secondLock = (index + 1) % resources;

        try {
            System.out.println("Thread " + index + " attempting to acquire lock " + firstLock);
```

```
            locks[firstLock].lock();
            System.out.println("Thread " + index + " acquired lock " + firstLock);

            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }

            System.out.println("Thread " + index + " attempting to acquire lock " + secondLock);
            locks[secondLock].lock();
            System.out.println("Thread " + index + " acquired lock " + secondLock);
        } finally {
            //Case where the number of resources is greater than the number of threads, where the deadlock will not occur
            locks[secondLock].unlock();
            locks[firstLock].unlock();
            System.out.println("Thread " + index + " released locks " + firstLock + " and " + secondLock);
        }
    }
}
```

Output :

```
(base) ➜ deadlock  /usr/bin/env /usr/lib/jvm/java-17-openjdk/bin/java -XX:+ShowCodeDetailsInExce
Enter number of resources: 3
Enter number of processes: 4
Thread 1 attempting to acquire lock 1
Thread 0 attempting to acquire lock 0
Thread 3 attempting to acquire lock 0
Thread 2 attempting to acquire lock 2
Thread 1 acquired lock 1
Thread 2 acquired lock 2
Thread 0 acquired lock 0
Thread 1 attempting to acquire lock 2
Thread 2 attempting to acquire lock 0
Thread 0 attempting to acquire lock 1
^C
```

When the resources are more than the process then the deadlock doesn't occur:

Output :

```
(base) ➜ deadlock  /usr/bin/env /usr/lib/jvm/java-17-openjdk/bin/java -XX:+ShowCodeDetailsInExceptionMessages
Enter number of resources: 4
Enter number of processes: 3
Thread 0 attempting to acquire lock 0
Thread 2 attempting to acquire lock 2
Thread 0 acquired lock 0
Thread 2 acquired lock 2
Thread 1 attempting to acquire lock 1
Thread 1 acquired lock 1
Thread 0 attempting to acquire lock 1
Thread 2 attempting to acquire lock 3
Thread 1 attempting to acquire lock 2
Thread 2 acquired lock 3
Thread 1 acquired lock 2
Thread 0 acquired lock 1
Thread 1 released locks 1 and 2
Thread 0 released locks 0 and 1
Thread 2 released locks 2 and 3
(base) ➜ deadlock
```