LAB-9

DEADLOCK DETECTION & AVOIDANCE

Prelab

1) no. of courses.

```
class Solution {
    public boolean canFinish(int numCourses,
    int[][] prerequisites) {

    ArrayList[] graph = new ArrayList[numCourses];
    int degree[] = new int[numCourses];
    Queue queue = new LinkedList();
    int count=0;
    for(int i=0; i<numCourses; i++)
        graph[i] = new ArrayList();
    for(int i=0; degree.length; i++)
    {
        if(degree[i]==0) {
            queue.add(i);
            count++;
        }
    }
```

```
while (queue.size() != 0) {
        int course = (int) queue.poll();
        for (int i=0; i<graph[course].size(); i++)
        {
            int pointer = (int) graph[course].get(i);
            degree[pointer] = --;
            if (degree[pointer] == 0) {
                queue.add(pointer);
            } count++;
        } }
    if (count == numCourses)
        return true;
    else
        return false;
    }
}
```

OUTPUT:

INPUT: 2
[[1,0]].

OUTPUT: true

3rd Lab

1) The Dining philosophers

```
class DiningPhilosophers {

    Private static final int N = 5;

    Private final List<Semaphore> forkSemaphores;
    Private final Semaphore sitting Semaphore = new
                                    Semaphore(N-1, true);

    Public DiningPhilosophers() {
        forkSemaphores Obj = new ArrayList<>();
        for(iN-i=0; i<N; ++i) {
            forkSemaphores.add(new Semaphore(1,true);
    } }

    Public void wantsToEat(int philosopher, Runnable
                                pickLeftFork, Runnable
                            pickRightFork,
    Runnable eat, Runnable putLeftFork, Runnable putRight
    Fork) throws InterruptedException {
```

```
sittingSemaphore.acquire();

Semaphore leftForkSemaphore = forkSemaphores.get
                                (Philosopher);
Semaphore rightForkSemaphore = forkSemaphore.get((
                                Philosopher+1)%N);

    leftForkSemaphore.acquire();
    rightForkSemaphore.acquire();

    pickLeftFork.run();
    pickRightFork.run();
    eat.run();
    putLeftFork.run();
    putRightFork.run();

    leftForkSemaphore.release();
    rightForkSemaphore.release();

    sittingSemaphore.release();
  }
}
```

RSTLab:

1) Package RSTLab;

Public class Problem3 {

Public static void main(String[] args) {

final String s = "Print";

final String s1 = "Scanner";

Thread t1 = new Thread() {

Public void run() {

synchronized(s) {

System.out.println("Desktop locked"

+ s);

try {

Thread.sleep(5000);

}

catch (InterruptedException e) {

}

}

synchronized(s1) {

System.out.println("Desktop locked"

+ s1);

}

```
Thread t2 = new Thread(){
    public void run(){
        synchronized(s1){
            System.out.println("Laptop locked "+s1);
            try{
                Thread.sleep(100);
            }
            catch(Exception e){
            }
            synchronized(s2){
            System.out.println("Laptop locked "+s2);
            }
        }
    }
};

t1.start();
t2.start();
}
}
```