Module Code	**:**	CT074-3-2-CCP
Intake Code	**:**	APD2F2311CS(DA)
Lecturer Name	**:**	TS.DR. MAYTHEM KAMAL ABBAS AL-ADILEE
Hand in Date	**:**	Tuesday, 09 July 2024, 11:59 PM
Tutorial No.	**:**	LAB 7

Student ID	**Student Name**
TP069510	Abdul Muhaimin Aman

Table of Contents

1.0 INTRODUCTION

Air traffic management is one of the essential and challenging aviation disciplines, as many processes must work in unison to guarantee the appropriate achievement of the aircraft's goals. This project mimics the functioning of Asia Pacific Airport which is a small airport with only one taxiway and the maximum number of planes that can occupy the airport's terrain at a particular time is three planes at the most. It records the possibility, schedule, and manner of an aircraft's arrival, functioning, refueling, and taking off at an airport without interference and within constraints.

Essentially, the features that are effectively modeled include passenger boarding and discharge, aircraft cleaning, and fueling exclusive of passengers. The model also raises a constrained and stressful context in which gates are scarce and a real emergency landing due to fuel exhaustion The model presents a rather stressing and realistic situation to the airport. The patterns of the arriving aircraft are random and that's why this model reflects the actual environment of the aviation industry.

Hence, this project seeks to enable proper understanding of waiting list, number of planes attended, and passengers held through measurements of waiting times, number of planes served, and passengers boarded among others. Written in Java, this simulation draws upon concurrent program paradigms of handling multiple processes adding to the comprehension of effective traffic control.

2.0 ASSUMPTIONS

- **Runway and Ground Capacity:**

  Landing as well as the taking off planes uses a single strip of that of the airport.

  At any given time, the airport cannot have more than three airplanes on the ground at a given time whether on a runway.

- **Aircraft Operations:**

  Each aircraft follows a sequential process: boarding and disembarking of passengers, getting to the gate, acquiring supplies, getting fuel, picking up passengers, going to the runway, and flying.

  Each of the steps of the process takes a specific amount of time and it is set and rigid for every aircraft that encounters it.

- **Passenger Handling:**

  They are disposed in a manner that either wants to board a compartment immediately or alight immediately from it.

  The passenger transport capacity for each of the airplanes is up to 50 persons.

- **Concurrent Processes:**

  People getting off the plane, and others boarding this flight, as well as cleaning of the aircraft and refilling of supplies can be conducted at the same instance but in different airplanes.

  Refueling is elitist and can only be conducted on one aircraft at a time because there is only one fueling truck.

- **Aircraft Arrival:**

  Time differences among arrivals of aircrafts vary randomly and can be 0, 1 or 2 seconds or any other near figure which has been generated randomly.

  There is no location for aircraft on the ground; any plane that arrives and all gates are occupied, must taxi around the airport until a gate is freed up.

- **Emergency Scenarios:**

  An intensive setting is added in which two planes are taxiing for landing while two gates are occupied, and a third plane with a low fuel supply needs to land urgently.

- **Sanity Checks and Statistics:**

  At the end of a simulation all gates should be looked to see that they are all empty before proceeding.

  Quantitative data shall be measured in terms of the maximum, average and minimum waiting time that was accorded to each aircraft and the numbers of planes that were served not to mention the number of passengers that were boarded.

3.0 BASIC REQUIREMENTS

3.1 LIST OF REQUIREMENTS MET

- **Single Runway:**

  The airport site has a single strip that accommodates all the aircrafts arriving and those going out of the airport.

- **Maximum Ground Capacity:**

  There must not be more than three airplanes on the airport territory or field, including the plane on the take-off strip.

- **Sequential Aircraft Operations:**

  Aircraft must follow a specific sequence: arrive at the runway, toot for a certain stand, park at the stand, let passengers de-board, restock food, water and fuel, let aboard new passengers, pull back from the gate, and taxi to the take-off runway.

- **Time Management:**

  Every process involving the aircraft is scheduled to take a particular amount of time.

- **No Waiting Area:**

  There are no facilities for the planes to park on the tarmac; should all gates be taken, arriving airplanes must circle in the air until a gate is vacated.

3.2 CONCEPTS IMPLEMENTED

3.2.0 ATOMIC STATEMENTS

Atomic statements guarantee that some operation is going to have a completion with no interference in the information maintaining coherency. Atomicity is implemented in the given code through the keyword synchronized, which means that the specified methods can be executed only by a single thread.

```
public synchronized void recordWaitTime(long waitTime) {
    waitTimes.add(waitTime);
}

public synchronized void recordServedAircraft() {
    aircraftServed++;
}

public synchronized void recordPassengersBoarded(int passengers) {
    passengersBoarded += passengers;
}
```

recordWaitTime: It synchronously appends to the list an entry of a wait time.
recordServedAircraft: Updates the counter of served aircraft with atomic manner.
recordPassengersBoarded: Increments the passengers boarded counter atomically.
Applying synchronized guarantees these updates to the shared resources are done in an atomic manner capable of handling race conditions.

3.2.1 SYNCHRONIZATION

The synchronized keyword is used to manage the access rights of the critical section and coordinate just one thread to perform just one section of code at the time of its execution.

```
public synchronized void requestEmergencyLanding() {
    emergencyLandingRequested = true;
}

public synchronized boolean isEmergencyLandingRequested() {
    return emergencyLandingRequested;
}
```

requestEmergencyLanding: This method is done in parallel to ensure a point of time that one thread may be setting the emergencyLandingRequested flag, where the other thread cannot access this flag at the time; making it safe.
isEmergencyLandingRequested: This is done in order to make sure that while one thread is for instance reading the value in emergencyLandingRequested flag, the other thread cannot modify this flag and hence provide a consistent flag value always.

3.2.2 SEMAPHORE

Semaphore restricts the number of threads that are accessing a shared resource at any given time.

```
private final Semaphore runwaySemaphore = new Semaphore(1, true);
private final Semaphore gateSemaphore = new Semaphore(MAX_AIRCRAFT_ON_GROUND - 1, true);
private final Semaphore refuelSemaphore = new Semaphore(1, true);
```

From the provided code, there are three common semaphores: for the runway, gates, and refueling. There are two semaphores for runway: runwaySemaphore for single plane at a time during landing/taking off acquire() operation is used to get the permission and release() is used to free the semaphore. This gate's semaphore, gateSemaphore, regulates the numbers at the gates, permitting not more than MaxAircraftOnGround – 1 plane at the gates. For similar use to the acquire() and release(), the refuel semaphore (refuelSemaphore) allows only one plane to refuel at a time. These semaphores effectively supervise the concurrency and sharing of the resources in the system.

4.0 ADDITIONAL REQUIREMENTS

4.1 LIST OF ADDITIONAL REQUIREMENTS MET

- Concurrent Passenger Operations:

  The three gates have passengers performing embark and disembark operations in parallel.

- Concurrent Refill Supplies and Cleaning:

  Aircraft refill supplies and cleanup is done in parallel with passenger operations.

- Exclusive Refueling:

  There is only one refueling truck available, so refueling must be done exclusively.

- Congested Scenario:

  Simulate a scenario in which two planes are waiting to land, and two gates are occupied; then a third plane will appear requesting an emergency landing because of low fuel, and this will have priority.

- Statistics and Sanity Checks:

  At the end of the simulation, all gates shall be empty. Collect and print statistics recording the maximum, average, and minimum waiting times for planes.

Count and print the total number of planes served along with the total number of passengers boarded.

4.2. CONCEPTS IMPLEMENTED

4.2.0 WAIT AND NOTIFY

These algorithms ensure that the planes will be made to wait until a space at the ground is available.

```
public void requestLanding(Plane plane) throws InterruptedException {
    synchronized (this) {
        while (aircraftOnGround >= MAX_AIRCRAFT_ON_GROUND && !plane.isEmergency()) {
            System.out.println("Thread-ATC: Plane " + plane.getId() + " requesting permission to land. Please wait and join the circle q
            wait();
        }
        aircraftOnGround++;
        System.out.println("Thread-ATC: Plane " + plane.getId() + " granted permission to land.");
    }

    runwaySemaphore.acquire();
    System.out.println("Thread-Plane-" + plane.getId() + ": is landing.");
    plane.sleepRandom();
    runwaySemaphore.release();
}

public void dock(Plane plane) throws InterruptedException {
    gateSemaphore.acquire();
    System.out.println("Thread-Plane-" + plane.getId() + ": is docking.");
    plane.sleepRandom();
    gateSemaphore.release();
}

public void refuel(Plane plane) throws InterruptedException {
    refuelSemaphore.acquire();
    System.out.println("Thread-Plane-" + plane.getId() + ": is refueling.");
    plane.sleepRandom();
    Thread supplyAndRefillThread = new Thread(new SupplyAndRefill(plane.getId()));
    supplyAndRefillThread.start();
    refuelSemaphore.release();
}

public void refuelEmergency(Plane plane) throws InterruptedException {
    refuelSemaphore.acquire();
    System.out.println("Thread-Plane-" + plane.getId() + ": is refueling EMERGENCY.");
    Thread supplyAndRefillThread = new Thread(new SupplyAndRefill(plane.getId()));
    supplyAndRefillThread.start();
    refuelSemaphore.release();
}

public void takeOff(Plane plane) throws InterruptedException {
    runwaySemaphore.acquire();
    System.out.println("Thread-Plane-" + plane.getId() + ": is taking off.");
    plane.sleepRandom();
    runwaySemaphore.release();

    synchronized (this) {
        aircraftOnGround--;
        System.out.println("Thread-ATC: Plane " + plane.getId() + " has taken off. Gates are available.");
        notifyAll();
    }
}
```

In the given Java code, wait () and notifyAll () is implemented for thread synchronization. It is found in method requestLanding(Plane plane) that the competing threads, representing the planes, are waiting while the aircraftOnGround exceeds a maximum limit (MAX_AIRCRAFT_ON_GROUND) and the plane is not an emergency. When the condition permits, a plane lands, increments aircraftOnGround, and notifies the waiting threads. In takeOff(Plane plane), an aircraft acquires a semaphore to use the runway and after taking off, releases it. It updates aircraftOnGround and notifies all waiting threads about available space, thus enabling coordinated landing and takeoff operations in a multithreaded environment.

4.2.1 THREAD CREATION AND MANAGEMENT

Threads are created and managed to replicate the behaviour of planes, passengers, and supply refilling processes.

```
public void refuelEmergency(Plane plane) throws InterruptedException {
    refuelSemaphore.acquire();
    System.out.println("Thread-Plane-" + plane.getId() + ": is refueling EMERGENCY.");
    Thread supplyAndRefillThread = new Thread(new SupplyAndRefill(plane.getId()));
    supplyAndRefillThread.start();
    refuelSemaphore.release();
}
```

In the refuelEmergency(Plane plane) method, the handling of emergency refueling operations is combined with thread creation and management. Probably after acquiring the refuelSemaphore that controls the access to refueling resources, the method logs that the plane is refueling under emergency conditions. It then creates a new thread, supplyAndRefillThread, using the Thread class and passes an instance of SupplyAndRefill with the ID of the plane. In this thread, supplyAndRefillThread.start() initially starts the refilling process to execute in parallel. At the same time, the main thread will release the semaphore so that other planes can start refueling in parallel. This solution uses resources effectively and provides responsive behavior during emergency handling in a multithreaded setting.

5.0 REQUIREMENTS NOT MET

5.1 CONGESTED SCENARIO

The congested scenario that I have designed does handle an emergency refuelling situation, however in the output the emergency landing occurs only when the gates are available.

6.0 REFERENCES

Daga, B. (2022, November 28). *What are Threads in Java? How to Create a Thread with Examples*. freeCodeCamp.org. https://www.freecodecamp.org/news/what-are-threads-in-java-how-to-create-one/

GeeksforGeeks. (2023, May 21). *Synchronization in Java*. GeeksforGeeks. https://www.geeksforgeeks.org/synchronization-in-java/

Metta, N. (2023, July 11). Atomic Operations in Java: Mastering thread safety and concurrency. *Medium*. https://naveen-metta.medium.com/atomic-operations-in-java-mastering-thread-safety-and-concurrency-7c3360ec0bc5

Pankaj. (2022, August 3). *Java Thread wait, notify and notifyAll Example*. DigitalOcean. https://www.digitalocean.com/community/tutorials/java-thread-wait-notify-and-notifyall-example

Simplilearn. (2023, February 22). *What is Semaphore in Java and its Use?* Simplilearn.com. https://www.simplilearn.com/what-is-semaphore-in-java-uses-article