INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT074-3-2

CONCURRENT PROGRAMMING

APD2F2109CS(DA) / APU2F2109SE / APU2F2109CS(DA) / APD2F2109CS / APD2F2109SE / APU2F2109CS

HAND OUT DATE: 11 MARCH 2022

HAND IN DATE: 3 JUNE 2022

WEIGHTAGE: 20%

INSTRUCTIONS TO CANDIDATES:

1. **Submit your assignment at the administrative counter.**
2. **Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing).**
3. **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**
4. **Cases of plagiarism will be penalized.**
5. **The assignment should be bound in an appropriate style (comb bound or stapled).**
6. **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**
7. **You must obtain 50% overall to pass this module.**
8. **This report is Part 1 of 2 for the assignment.**

1. Basic Requirements Met

- There is only 1 runway for all planes to land and depart.

```
if (airport.getRunway().tryLock(arriveTime + fuelTime - System.currentTimeMillis() - durationToAnotherAirport - decisionTime, TimeUnit.MILLISECONDS)) {

    System.out.println(this.getAircraftCodeName() + " is using the runway for landing after assigned to " + assignedGate.get().getGateCodeName() + ".");
    Thread.sleep(landingTime);

    System.out.println(this.getAircraftCodeName() + " has completed landing in " + landingTime + " milliseconds.");
    if (!status.compareAndSet(Status.LANDING, Status.DOCKING)) {
        System.out.println(this.getAircraftCodeName() + "'s status is illegal!");
        rescue();
        airport.getRunway().unlock();
    }
} else {
    if (assignedGate.get() != null) {
        assignedGate.get().notifyAll();
    }
    airport.getTraffic().getRoadToIntersectionFromRunway().release();
    return;
}
```

Figure 1. Application of Lock for Airport Runway

- Ensure that the aircraft does not collide with another aircraft on the runway or gates.
- Once an aircraft obtains permission to land, it should land on the runway, coast to the assigned gate, dock to the gate, allow passengers to disembark, refill supplies and fuel, receive new passengers, undock, coast to the assigned runway and take-off.
- Each step should take some time.
- A congested scenario should be simulated where planes are waiting to land while the 2 gates are occupied.
- As the airport is small, there is no waiting area on the ground for the planes to wait for a gate to become available.

```
Aircraft No.1 is using the runway for landing after assigned to Gate B.
Aircraft No.1 has completed landing in 156 milliseconds.
Aircraft No.1 has freed the runway access after landing.
Aircraft No.1 is on the road to intersection.
Road to Gate B from intersection is free.
Aircraft No.1 is using the intersection.
Aircraft No.1 is on the road to Gate B.
Aircraft No.1 has docked to Gate B in 126 milliseconds.
Aircraft No.1 allows passenger to disembark. There are a total of 18 passengers on board.
Aircraft No.1 ongoing activity Refill fuel
```

Figure 2. Result Snippet

1.1 Semaphore

With the use of a counter, a Semaphore regulates permission to a shared resource. Access is permitted if the counter is larger than zero. If the value is zero, access is denied. Permits that enable permission to the shared resource are counted by the counter. As a result, a thread must obtain permission from the semaphore in order to access the resource (GeeksforGeeks, 2018).

```
traffic.setRoadToNearestGateFromRunway(new Semaphore(roadLength, true));

traffic.setRoadToGateFromIntersection(new Semaphore(roadLength, true));

traffic.setRoadToIntersectionFromRunway(new Semaphore(roadLength, true));

traffic.setRoadToStandbyFromIntersection(new Semaphore(roadLength, true));

traffic.setRoadToStandbyFromNearestGate(new Semaphore(roadLength, true));

traffic.setStandbyTakeOff(new Semaphore(standbyLength, true));
```

Figure 3. Semaphore for Road Permission

1.2 Synchronization

Threads primarily communicate by sharing permissions to areas and the objects referenced by reference fields. This method of communication is very efficient, but it allows for two types of errors: thread intervention and memory consistency failures. Synchronization is the tool required to avoid these errors (Oracle, n.d.).

```
public synchronized void takingOff() {
    try {
        int takeOffTime = (rnd.nextInt(3 * minutesToMilliseconds) + 1 * minutesToMilliseconds);
        Thread.sleep(takeOffTime);
        System.out.println(this.getAircraftCodeName() + " took off in " + takeOffTime + " milliseconds.");
        stat.updateAircraftStats(arriveTime, System.currentTimeMillis());
        if (!status.compareAndSet(Status.TAKEOFF, Status.LEFT)) {
            System.out.println(this.getAircraftCodeName() + "'s status is illegal!");
        }
    } catch (Exception e) {
        e.printStackTrace();
        return;
    } finally {
        System.out.println(this.getAircraftCodeName() + " has freed the runway access after taking off.");
        airport.getRunway().unlock();

    }
}
```

Figure 4. Synchronized for Aircraft Taking Off

1.3 Atomic Variable

When concurrency is used in multithreading, the sharable unit is usually a problem. A shared entity, such as a mutable object or attribute, may be changed, resulting in program or database inconsistency. As a result, it's critical to deal with the sharable entity while it's being accessed at the same time. In this case, an atomic variable could be one of the options (GeeksforGeeks, 2021).

```
public AtomicBoolean getIsOperate() {
    return isOperate;
}
```

Figure 5. AtomicBoolean for getIsOperate Status

1.4 Blocking Queue

In addition to queueing, the BlockingQueue interface provides flow control by incorporating blockage if either BlockingQueue is empty or full. A thread attempting to enqueue a component in a full queue will be blocked until another thread clears the queue, whether by dequeuing one or more components or by clearing the queue entirely. It also prevents a thread from deleting from an empty queue until another thread inserts an item. A null value is not accepted by BlockingQueue. NullPointerException is thrown if a null item is tried to be enqueued (GeeksforGeeks, 2021).

```
public BlockingQueue<Aircraft> getNormalQueue() {
    return normalQueue;
}
```

Figure 6. Blocking Queue for getting Normal Queue Value

2. *Additional Requirements Met*

These events should happen concurrently:

- Passengers disembarking/embarking

```
LinkedList<Passenger> embarkingPassengers = new LinkedList();
int passengerTotalCount = new Random().nextInt(60) + 1;
for (int i = 0; i < passengerTotalCount; i++) {
    Passenger embarkingPassenger = new Passenger(newAircraft, 'E');
    embarkingPassengers.add(embarkingPassenger);

    Thread embarkingPassengerThread = new Thread(embarkingPassenger);
    embarkingPassengerThread.start();
}
lounge.getWaitingPassengersTable().put("" + aircraftCounter, embarkingPassengers);
aircrafts[aircraftCounter - 1] = newAircraft;
aircraftThreads[aircraftCounter - 1] = new Thread(newAircraft);
aircraftThreads[aircraftCounter - 1].start();
```

Figure 7. Code Snippet for Customers Embarking

```
System.out.println(this.getAircraftCodeName() + " allows passenger to disembark. There are a total of " + passengersOnBoard.size() + " passengers on board.");
if (allowDisembark.compareAndSet(false, true)) {

    for (Passenger p : passengersOnBoard) {
        synchronized (p) {
            p.notify();
        }
    }
    //Wait until passengers are all disembarked
    while (passengersOnBoard.size() != 0) {

    }
    System.out.println(this.getAircraftCodeName() + " has all the passengers disembarked.");
}
```

Figure 8. Code Snippet for Customer Disembarking

- Refill supplies and cleaning of aircraft

```
Thread cleanCabin = new Thread(new Activity(this.getAircraftCodeName(), "Clean cabin", 21, 10));
Thread refillSupplies = new Thread(new Activity(this.getAircraftCodeName(), "Refill supplies", 21, 10));
cleanCabin.start();
refillSupplies.start();

cleanCabin.join();
refillSupplies.join();
```

Figure 9. Code Snippet for Cleaning Cabin and Supply Refilling

As there is only 1 refueling truck, this event should happen exclusively:

- Refueling of aircraft

```
Thread refillFuel = new Thread(new Activity(this.getAircraftCodeName(), "Refill fuel", 51, 10));
refillFuel.start();
```

Figure 10. Code Snippet for Refueling Fuel

The Statistics

At the end of the simulation, i.e., when all planes have left the airport, the ATC manager

should do some sanity checks of the airport and print out some statistics on the run. The result

of the sanity checks must be printed. You must:

- Check that all gates are indeed empty.
- Print out statistics on
- Maximum/Average/Minimum waiting time for a plane.
- Number of planes served/Passengers boarded.

```
System.out.println("Aircrafts Served = " + totalAircraftServed);
System.out.println("Aircrafts Failed = " + totalAircraftFailed);
System.out.println("Passengers = " + totalPassengers);
System.out.println("Passengers Embarked = " + totalPassengersEmbarked);
System.out.println("Passengers Disembarked = " + totalPassengersDisembarked);
System.out.println("Passengers Rejected = " + totalPassengersRejected);
```

Figure 11. Code Snippet for Statistics

3. Requirements Not Met

None

4. References

GeeksforGeeks. (2018, Dec 10). *Semaphore in Java*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/semaphore-in-java/

GeeksforGeeks. (2021, Jul 10). *Atomic Variables in Java with Examples*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/atomic-variables-in-java-with-examples/

GeeksforGeeks. (2021, Sep 18). *BlockingQueue Interface in Java*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/blockingqueue-interface-in-java/

Oracle. (n.d.). *Synchronization*. Retrieved from The Java™ Tutorials: https://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html