INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT074-3-2-CCP-022022-ZAB

CONCURRENT PROGRAMMING

APU2F2109CS(DA)

HAND OUT DATE: 11 MARCH 2022

HAND IN DATE: 3 JUNE 2022

WEIGHTAGE: 25%

STUDENT NAME: RYAN MARTIN

TP NUMBER: TP058091

System Requirements

All basic requirements of the program as specified by the assignment question document, have been met by the student's program. These requirements includes:

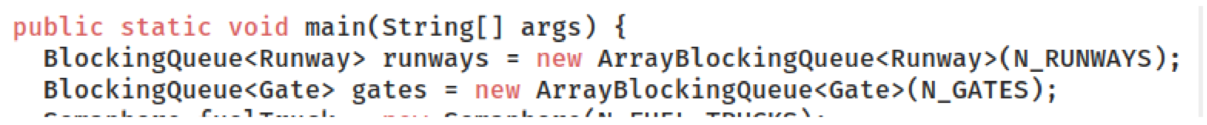
1. Having only 1 runway,
2. No collision between aircrafts on either the runway or gates,
3. Once permitted the plane will land, head to the assigned gate, dock to the gate and allow passengers to disembark, refill supplies and fuel, accept new passengers, and finally requesting for and take off,
4. Each step should take time to execute,
5. A congested scenario should be simulated,
6. Planes cannot wait on the ground

In addition, all the additional requirements as specified by the assignment question document have also been met, which includes:

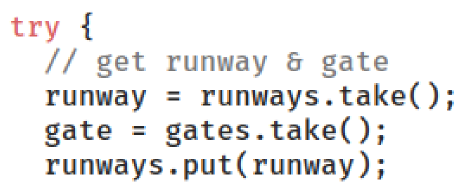
1. Passengers embarking/disembarking & plane refuelling should happen concurrently,
2. Having only 1 refuel truck available at any time

Concurrency Concepts Used

Blocking Queue



```
public static void main(String[] args) {
  BlockingQueue<Runway> runways = new ArrayBlockingQueue<Runway>(N_RUNWAYS);
  BlockingQueue<Gate> gates = new ArrayBlockingQueue<Gate>(N_GATES);
```



```
try {
  // get runway & gate
  runway = runways.take();
  gate = gates.take();
  runways.put(runway);
```

A blocking queue is a variant of the queue data type that blocks operations on itself when the queue is either full or empty. In Java, this data type is implemented as an interface called BlockingQueue. It can then be used by using the classes that implement this interface such as the ArrayBlockingQueue or LinkedBlockingQueue classes. These classes as well as the interface can be accessed from the java.util.concurrent package.

In the student's program, the runway as well as the gates are implemented as blocking queues. When a plane wants to land, it will call the take() method on the runway queue, and once it has landed and taken a gate, it will return the taken runway back into the queue. If there is no runway available in the queue (meaning it is in use by another thread), the thread will be blocked until it is available again.

Semaphore

```
Semaphore fuelTruck = new Semaphore(N_FUEL_TRUCKS);
```

```
try {
  // get semaphore here
  truck.acquire();
  System.out.println(
    String.format("[plane %d] refuel truck comes & refuels plane",
                  planeNumber));
  Thread.sleep(5000);
  truck.release();
```

A semaphore is an abstract data type that provides a way to manage shared resources in a concurrent program. Unlike locks, semaphores allow for multiple threads to access a shared resource, and it holds a count variable that denotes the number of threads that can get access to a shared resource. A semaphore can be acquired until this count reaches 0, which will block future access to a shared resource until a semaphore is released by the other threads.

In the student's program, a semaphore is used for the fuel truck. Once a plane needs refuelling, it will acquire the semaphore using the acquire() method, blocking all other threads from accessing the fuel truck. Once it's done, it will release the semaphore by calling the release() method.

Synchronised Keyword

```
  // send stats to the Statistics class
  synchronized (stats) {
    stats.addWaitingTime(planeNumber, waitingTime);
  }
```

The synchronised keyword has already been explained in the program proposal, so it will not be discussed much again in this document. It is an implicit lock, and in the program it is used to control the usage of the stats property, which is a shared resource, to only 1 thread at a time. The stats property is used to collect statistics of the program to be printed at the end.

Sample Output

Here is a sample output of the program.

```
[main thread] airport simulation started...

[plane 1] arrives at airport & requests for landing
[plane 1] lands on runway & docks at gate 1
[plane 2] arrives at airport & requests for landing
[plane 2] lands on runway & docks at gate 2
[plane 3] arrives at airport & requests for landing
[plane 1] passengers gets off plane to gate 1
[plane 2] passengers gets off plane to gate 2
[plane 4] arrives at airport & requests for landing
[plane 5] arrives at airport & requests for landing
[plane 1] staff cleans plane & refills supplies
[plane 6] arrives at airport & requests for landing
[plane 2] staff cleans plane & refills supplies
[plane 1] refuel truck comes & refuels plane
[plane 1] refuel truck finishes refuelling plane
[plane 1] passengers gets on plane from gate 1
[plane 2] refuel truck comes & refuels plane
[plane 1] requests for take-off
[plane 2] refuel truck finishes refuelling plane
[plane 1] takes-off from the runway
[plane 2] passengers gets on plane from gate 2
[plane 3] lands on runway & docks at gate 1
[plane 2] requests for take-off
[plane 3] passengers gets off plane to gate 1
[plane 2] takes-off from the runway
[plane 3] staff cleans plane & refills supplies
[plane 4] lands on runway & docks at gate 2
[plane 3] refuel truck comes & refuels plane
[plane 4] passengers gets off plane to gate 2
[plane 3] refuel truck finishes refuelling plane
[plane 4] staff cleans plane & refills supplies
[plane 4] refuel truck comes & refuels plane
[plane 3] passengers gets on plane from gate 1
[plane 3] requests for take-off
[plane 4] refuel truck finishes refuelling plane
[plane 3] takes-off from the runway
[plane 4] passengers gets on plane from gate 2
[plane 5] lands on runway & docks at gate 1
[plane 4] requests for take-off
[plane 5] passengers gets off plane to gate 1
```

```
[plane 4] takes-off from the runway
[plane 5] staff cleans plane & refills supplies
[plane 6] lands on runway & docks at gate 2
[plane 5] refuel truck comes & refuels plane
[plane 6] passengers gets off plane to gate 2
[plane 6] staff cleans plane & refills supplies
[plane 5] refuel truck finishes refuelling plane
[plane 6] refuel truck comes & refuels plane
[plane 5] passengers gets on plane from gate 1
[plane 5] requests for take-off
[plane 6] refuel truck finishes refuelling plane
[plane 5] takes-off from the runway
[plane 6] passengers gets on plane from gate 2
[plane 6] requests for take-off
[plane 6] takes-off from the runway
[main thread] airport simulation ended...

simulation statistics:
longest waiting time: 53.00 seconds [plane 5]
shortest waiting time: 0.00 seconds [plane 1]
average waiting time: 26.27 seconds
planes served: 6
```