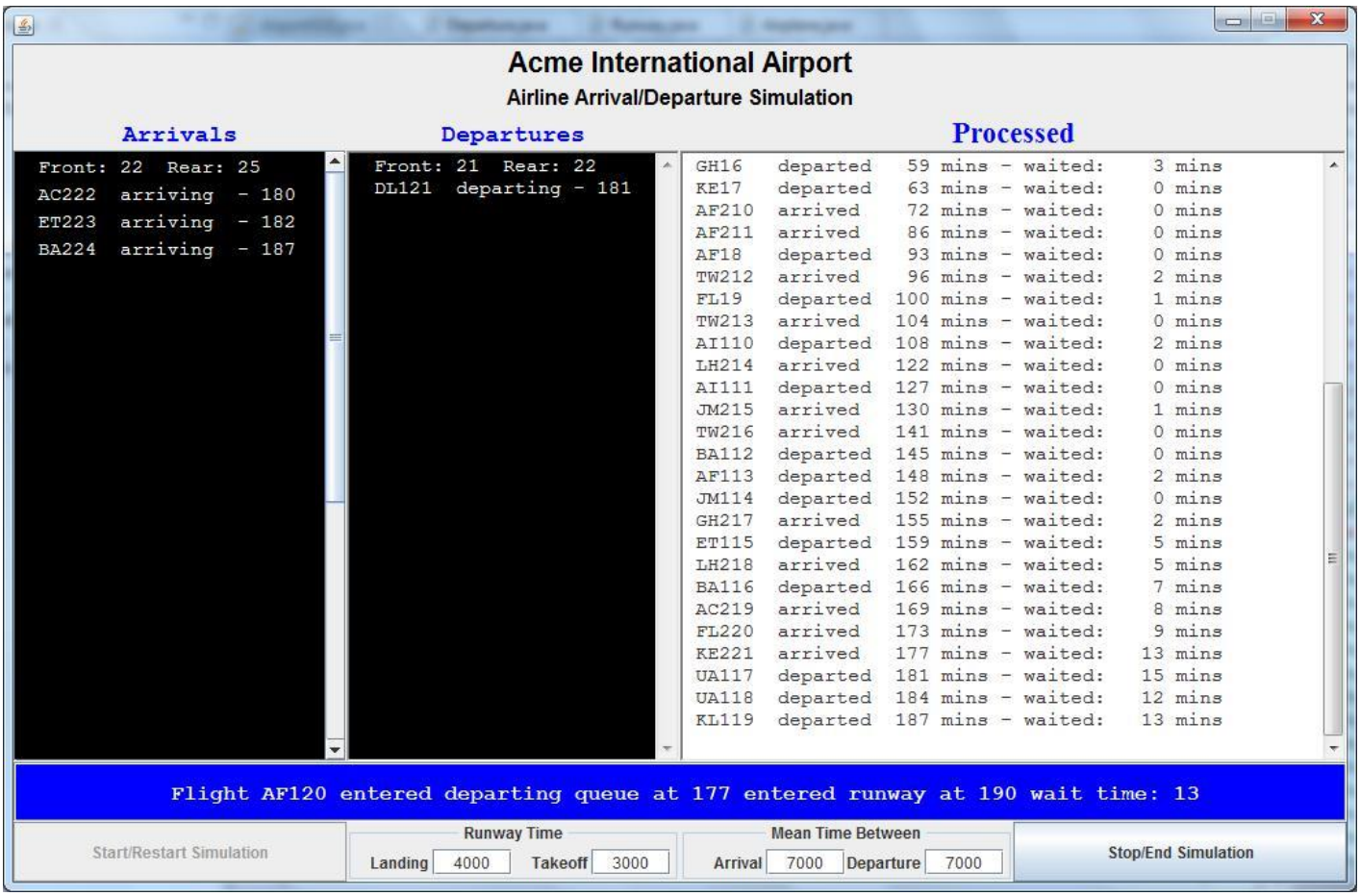
# CSCI212 Program #3 - Airport Simulation

Write a simulation for a small airport that has one runway. There will be a queue of planes waiting to land and a queue of planes waiting to take off. Only one plane can use the runway at a time. The program should run for 5 to 10 minutes, but use seconds to simulate the minutes used for takeoff, landing and the mean time for each event.

- All takeoffs take the same amount of time and all landings take the same amount of time (though these two times may be different).
- Planes arrive for landing at random times, based on the mean even time of (4 minutes).
- Planes depart (take off) at random times, based on the mean even time (5 minutes).
- Arrivals have priority over departures. Planes arriving for landing have a random amount of fuel and they will crash if they do not land before they run out of fuel.
- Planes in the takeoff queue are ordered by time of arrival in the queue (FIFO).
- Planes in the landing queue are ordered by time of arrival in the queue (FIFO).

The output of the program will be something like the following:




**Text-Based Output:**

```
Minute: 0 - Added flight FL10 to departure Queue
Random wait time before next departure: 7 mins
Minute: 0 - Added flight LH20 to arrival Queue
Minute: 0 - Flight LH20 cleared for landing - Entered Queue at 0 - waited 0 mins
Random wait time before next arrival: 0 mins
Minute: 0 - Added flight AC21 to arrival Queue
Random wait time before next arrival: 4 mins
Minute: 2 - Flight AC21 cleared for landing - Entered Queue at 0 - waited 2 mins
Minute: 4 - Flight FL10 cleared for takeoff - Entered Queue at 0 - waited 4 mins
Minute: 4 - Added flight BA22 to arrival Queue
Random wait time before next arrival: 7 mins
Minute: 6 - Flight BA22 cleared for landing - Entered Queue at 4 - waited 2 mins
Minute: 7 - Added flight KL11 to departure Queue
Random wait time before next departure: 0 mins
Minute: 7 - Added flight AF12 to departure Queue
Random wait time before next departure: 8 mins
Minute: 8 - Flight KL11 cleared for takeoff - Entered Queue at 7 - waited 1 mins
Minute: 10 - Flight AF12 cleared for takeoff - Entered Queue at 7 - waited 3 mins
Minute: 12 - Added flight AF23 to arrival Queue
Random wait time before next arrival: 9 mins
Minute: 12 - Flight AF23 cleared for landing - Entered Queue at 12 - waited 0 mins
Minute: 15 - Added flight JM13 to departure Queue
Random wait time before next departure: 2 mins
Minute: 15 - Flight JM13 cleared for takeoff - Entered Queue at 15 - waited 0 mins
Minute: 17 - Added flight JM14 to departure Queue
Random wait time before next departure: 7 mins
Minute: 17 - Flight JM14 cleared for takeoff - Entered Queue at 17 - waited 0 mins
Minute: 21 - Added flight ET24 to arrival Queue
```

**Exponential distribution** (e.g. time until next event of a Poisson process)
With rate $\lambda = lim_{\Delta t \to 0}(\#\ of\ events\ in\ \Delta t)\ /\ \Delta t$ : $f_x(x) = \lambda e^{-\lambda x}\ for\ x \geq 0\ (0\ otherwise)$

**Poisson distribution** - a function that calculates the distribution of times between arrivals.

Let **x** be any time between arrivals. Then F(x), the probability that the time until the next event will be at least x minutes from now, is given by $F(x) = \exp(-x/meanEventTime)$

*1 – randomDouble = exp(–timeTillNext / meanEventTime)*

*log (1 – randomDouble) = –timeTillNext / meanEventTime*

*timeTillNext = – meanEventTime * log (1 –randomDouble)*

**Written in Java:**

*timeTillNext* = (**int**)Math.*round* (-*meanEventTime* * Math.*log* (1 - randomDouble));

**Threads:**

```
class MyThread extends Thread {

   MyThread() {
      …
   }

   public void run() {
      …
      try {
         sleep(5000); // stop for 5 seconds
      }
      catch(InterruptedException i){
         System.out.println("Exception");

      }
   }
}
```

```
MyThread theThread = new MyThread();

theThread.start();


while(theThread.isAlive(){

}
```

Methods in the Queue class should include the **synchronized** keyword.

```
public synchronized T dequeue() throws QueueEmptyException{
   …
}
```

| **Airline** | |
|---|---|
| -flightID : String | Plane's id tag  example: AA204 |
| -entered: long | Time the plane entered the queue |
| -exited: long | Time the plane exited the queue |
| +Airline(String, long) | Constructor requires the id & time entered |
| +getID() : String | Accessor for the id |
| +getEntered(): long | Accessor for the time entered |
| +setEntered(long ) | Mutator for the time entered |
| +setExited(long) | Mutator for the time exited |
| +toString() : String | A string representation of the Airline |

| **Thread** | |
|---|---|
| | |
| +interrupt() : void | Send an interrupt to the thread |
| +interrupted() : boolean | Check to see if thread was interrupted |
| +sleep(long) : void | Causes the thread to sleep for milliseconds |
| +isAlive() : boolean | Check to see if thread is active (running) |
| +join() : void | Waits for the thread to die |
| +start() : start | Starts the thread – calls run method |
| +**run**() : void | The body of the thread (***override***) |

| **Arrival** | |
|---|---|
| -queue : Queue<Airline> | Arrival queue |
| -time : int | Landing duration (sleep time) |
| -running : boolean | Used by main program to stop this thread |
| +Arrival(int) | Constructor requires the landing duration |
| +getQueue() : Queue<Airline> | Accessor for the arrival queue |
| +getTime() : int | Accessor for the landing duration time |
| +stopRunning() : void | Change the running state to false |
| +toString() : String | A string representation of the Arrival |
| +**run**() : void | Started by the ***start*** method, loop until main program calls the ***stopRunning*** method |

| **Thread** | |
|---|---|
| | |
| +interrupt() : void | Send an interrupt to the thread |
| +interrupted() : boolean | Check to see if thread was interrupted |
| +sleep(long) : void | Causes the thread to sleep for milliseconds |
| +isAlive() : boolean | Check to see if thread is active (running) |
| +join() : void | Waits for the thread to die |
| +start() : start | Starts the thread – calls run method |
| +**run**() : void | The body of the thread (***override***) |

| **Departure** | |
|---|---|
| -queue : Queue<Airline> | Arrival queue |
| -time : int | Takeoff duration (sleep time) |
| -running : boolean | Used by main program to stop this thread |
| +Departure(int) | Constructor requires the takeoff duration |
| +getQueue() : Queue<Airline> | Accessor for the arrival queue |
| +getTime() : int | Accessor for the mean time between arrivals |
| +stopRunning() : void | Change the running state to false |
| +toString() : String | A string representation of the Departure |
| +**run**() : void | Started by the ***start*** method, loop until main program calls the ***stopRunning*** method |

| **Thread** | |
|---|---|
| | |
| +interrupt() : void | Send an interrupt to the thread |
| +interrupted() : boolean | Check to see if thread was interrupted |
| +sleep(long) : void | Causes the thread to sleep for milliseconds |
| +isAlive() : boolean | Check to see if thread is active (running) |
| +join() : void | Waits for the thread to die |
| +start() : start | Starts the thread – calls run method |
| +**run**() : void | The body of the thread (***override***) |

| **Runway** | |
|---|---|
| -arrival : Arrival | A reference to the arrival thread |
| -departure : Departure | A reference to the departure thread |
| -running : boolean | Used by main program to stop this |
| +Runway(Arrival, Departure) | Constructor needs arrival & departure info |
| +stopRunning() : void | Change the running state to false |
| +toString() : String | A string representation of the Runway |
| +**run**() : void | Started by the ***start*** method, loop until main program calls the ***stopRunning*** method |

# The Basic Outline for the Program

```
public class Arrival extends Thread{
   private boolean running = true;

   public void run() {
      while(running){

            ....

      }
   }

   public void stopRunning(){
      running = false;
   }
}

public class Departure extends Thread{
   private boolean running = true;

   public void run() {
      while(running){

            ....

      }
   }

   public void stopRunning(){
      running = false;
   }
}

public class Runway extends Thread{
   private boolean running = true;

   public void run() {
      while(running){

            ....

      }
   }

   public void stopRunning(){
      running = false;
   }
}
```

```
public class Program3

   public void startSimulation(long time) {

      //TODO: Start each thread

      //Loop - run simulation for specified time
      while(System.currentTimeMillis() < startTime + simulationTime){

      }

      //TODO: stop the loop in each thread – call the stopRunning method


      //TODO: interrupt each thread – method from Thread class


      try {

         //TODO: wait for each thread to die – method from Thread class

      } catch (InterruptedException e){
         e.printStackTrace();
      }
   }

   public static void main(String[] args) {

      /* TODO:

         Input the length of time to run the simulation (in minutes)
         Create an instance of Program3
         Call the startSimulation method and pass the time to it
         Loop while the Arrival or Departure threads are alive
         Stop the Runway thread from running
      */
   }
}
```