Peter Rodeheffer

CSCI 1524-01

Classes:

**Airport.java**

**Description:**

Airport will simulate an airport with only one runway. It will have a queue of planes waiting to take off and a queue of planes waiting to land. Only one plane can use the runway at a time. Every runway use will take same amount of time based on input but a landing and take off can be different. Planes will arrive in each queue randomly, but each will have a probability. Landings have higher priority than take offs. A clock will track the time passed in minutes.

**Fields**:

* int landTime: time a plane takes to land, input
* int takeOffTime: time a plane takes to take off, input
* double landingProb: probability of landing plane arriving, 0 < x < 1
* double takeOffProb: probability of take off plane arriving, 0 < x < 1
* int fuelTime: amount of time a plane can stay in landing queue before crashing
* int simulationTime: how many minutes to run the simulation for

**Methods:**

* Public static void main(String[] args):
  + the main method that will run methods for getting inputs and initiate the simulation.
* private static void inputData()
  + prompts the user to input data for the class level fields
  + **Preconditions**: called by main method
  + **Postconditions**: all class level fields should have updated values
* Private static void outputData(int takeOffs, int landings, int crashes, int takeOffTotalTime, int landtotalTime)
  + Outputs the total number of crashes, landings, take offs, and the average time in each queue after the simulation.
  + **Parameters**
    - Int takeOffs: number planes that took off
    - Int landings: number of planes that landed
    - Int crashes: number of planes that crashed
    - Int takeOffTotalTime: total time spent in take off queue
    - Int landTotalTime: total time spent in landing queue
  + **Preconditions:** the simulation has been run to completion
  + **Postconditions:** data has been output to the user.
* Private static void runSimulation():
  + This method will contain the simulation loop for the runway. It instantiates the Runway and creates a landing queue and take off queue. It tracks the number of planes that successfully land, successfully take off, the number of crashes, and the total time spent in each queue respectively. It calls displayOutput() at the end of the simulation. Each run through the loop is one minute. Other than the first run, each run starts by updating the runway and then checking if it is clear. Then, each queue checks for newly arriving planes. If the runway is empty, the landing queue is checked. If it isn’t empty, planes are dequeued until one is found that hasn’t crashed or the landing queue is empty. If one is found, it is added to the runway. If the runway is still empty and the take off queue isn’t empty, a plane is then dequeued and added to the runway. Current time is then incremented by 1 and the loop restarts. After the simulation time is completed, the landing queue is check for any additional crashes before outputData() is called.
  + **Preconditions:** inputData() has been called. landProb and takeOffProb are greater than 0 but less than 1. All parameters are positive values.
  + **Postconditions:** takeOffs, landings, crashes, totalTakeOffTime, and totalLAndTime are used to call outputData().
  + **Fields:**
    - Int takeOffs: number of planes that took off during the simulation
    - Int landings: number of planes that landed during the simulation
    - Int crashes: number of planes that crashed during the simulation
    - int totalTakeOffTime total time planes spent in take off queue
    - int totalLandTime total time planes spent in landing queue
    - int currentTime how many minutes have passed so far
    - LinkedQueue landingQueue: the Queue object that represents the landing queue
    - LinkedQueue takeOffQueue: the Queue object that represents the take off queue
    - Random rand random generator object for simulating the chance of arriving planes in each queue each minute
    - Runway myRunway runway object to track the runway status

**Runway.java**

**Description:**

Runway should simulate a runway. It should store an Airplane object in runwayPlane when a plane is on the runway, and null when the runway is clear. It should also store the time the plane arrived on the runway and the time needed before it leaves.

**Fields:**

* + Airplane runwayPlane: represents a plane currently on the runway
  + int runwayArrivalTime time a plane arrived on runway
  + int timeNeeded time a plane needs to leave the runway

**Methods:**

* Public Runway()
  + Constructor that instantiates runwayPlane to null, timeNeeded to 0, and runwayArrivalTime to 0
  + **Preconditions**: None
  + **Postconditions:** object has been initiated
* Public void addToRunway(Airplane thisPlane, int currentTime, int runwayTime)
  + adds a provided plane to the runwayPlane, updates timeNeeded to runwayTime, and updates runwayArrivalTime to currentTime
  + **Parameters**
    - thisPlane: the plane to store in runwayPlane
    - currentTime: the time the plane is added to the runway, stored in runwayArrivalTime
    - runwayTime: time the plane needs on the runway, stored in timeNeeded
  + **Preconditions:** Runway is empty
  + **Postconditions:** runwayPlane, runwayArrivalTime, timeNeeded are updated to new values
* public Boolean isEmpty()
  + checks if the runway is empty by seeing if runwayPlane is null
  + **preconditions:** constructor has been called
  + **postconditions**: none
  + **return:** true if no planes on runway (runwayPlane is null), false if there are
* private void removeRunway()
  + sets runwayPlane to null
  + **precondition:** checkRunway() has been called
  + **postcondition:** the runway is now empty, runwayPlane is null
* public void checkRunway(int currentTime)
  + checks if the plane on the runway has had enough time to take off/land by comparing the result of currentTime - runwayArrivalTime to timeNeeded. If the two numbers are equal, calls removeRunway()
  + **parameters**
    - int currentTime: the current time of the simulation
  + **precondition:** the runway isn’t empty
  + **postcondition:** Runway is now empty or none

**Airplane.java**

**Description**:

Airplane should represent an airplane object. It should contain the planes queue arrival time, a getter for the arrival time, and method for checking if the plane has crashed based on provided fuel time/current time

* **Fields**
  + Int arrivalTime: the time that the plane arrived in the queue

**Methods:**

* Public Airplane(int currentTime)
  + Constructor that sets arrivalTime to the provided value, which is the current time
  + **Preconditions**: None
  + **Postconditions**: fields have been set and object initialized
* Public Boolean hasCrashed(int currentTime, int fuelTime)
  + Checks if the plane has crashed by checking if arrivalTime and fuelTime added together are less than currentTime
  + **Parameters**
    - currentTime: the currentTime of the simulation
    - fuelTime: time the plane can be in queue without crashing
  + **Preconditions**: airplane was in landing queue
  + **Postconditions:** plane can be removed from queue and crashes can be incremented by 1
  + **Return:** true if plane has crashed, false if not
* Public int getArrivalTime()
  + Returns arrivalTime
  + **Preconditions:** constructor called
  + **Postconditions**: none
  + **Return:** arrivalTime

UML Diagrams

Airport.java

-landTime: int

-takeOffTime: int

-landingProb: double

-takeOffProb: double

-fuelTime: int

-simulationTime: int

+main(): void

-inputData(): void

-runSimulation(): void

-outputData(takeOffs: int; landings: int; crashes: int; takeOffTimeTotal: int; landingTimeTotal: int): void

Runway.java

-runwayPlane: Airplane

-runwayArrivalTime: int

-timeNeeded: int

+Runway()

+isEmpty(): Boolean

+addToRunway(thisAirplane: Airplane; currentTime: int; runwayTime: int): void

-removeRunway(): void

+checkRunway(currentTime: int): void

Airplane.java

-arrivalTime: int

+Airplane(currentTime: int)

+getArrivalTime(): int

+hasCrashed(currentTime: int; fuelTime: int): boolean