```
1  import java.util.*;
2  import java.util.Scanner;
3  /**
4   *
5   * Simulate -- a mathematical model to determine optimal arrival times in
    airports
6   *
7   * @author Shiraz Johnson
8   *
9   */
10 public class Simulate{
11     Customer customer; //points to Customer class//
12 
13     double[] lambdaArray = {0.1, 0.1, 0.1666666667, 0.2666666667, 1, 1.06
    6666667, 2.433333333,
14             3.166666667, 3.166666667, 4.533333333, 3.166666667, 3.1666666
    67, 2.433333333, 1.066666667,
15             1, 0.2666666667, 0.2666666667, 0.2, 0.2666666667, 0.166666666
    7, 0.9, 0.9, 2.266666667,
16             2.266666667, 2.266666667, 2.266666667, 1, 1, 0.3333333333, 0.
    3333333333, 1, 1, 2.266666667,
17             2.266666667, 2.266666667, 2.266666667, 0.9, 0.9, 0.1666666667
    , 0.1666666667, 0.1, 0.1, 0.1,
18             0.1,0.1666666667, 0.1666666667, 0.9, 0.9, 2.266666667, 2.2666
    66667, 2.266666667, 2.366666667,
19             1, 1.066666667, 0.3333333333, 1.066666667, 1, 2.366666667, 2.
    266666667, 2.266666667,
20             2.266666667, 0.9, 0.9, 0.2666666667, 0.2666666667, 0.26666666
    67, 0.2666666667, 0.9, 0.9,
21             2.266666667, 2.266666667, 2.266666667, 2.266666667, 0.9, 0.9,
     0.2666666667, 0.2666666667,
22             0.2666666667, 0.3666666667, 1, 1.066666667, 2.433333333, 3.16
    6666667, 3.166666667, 4.533333333,
23             3.166666667, 3.166666667, 2.433333333, 1.066666667, 1, 0.2666
    666667, 0.1666666667, 0.1, 0.2,
24             0.1, 0.1666666667, 0.1666666667, 0.9, 0.9, 2.266666667, 2.266
    666667, 2.266666667, 2.266666667,
25             0.9, 0.9, 0.1666666667, 0.2666666667, 0.2, 0.3666666667, 0.26
    66666667, 1.066666667, 1.066666667,
26             3.166666667, 3.166666667, 4.533333333, 4.533333333, 3.1666666
    67, 3.166666667, 1.066666667,
27             1.066666667, 0.2666666667, 0.2666666667, 0.1, 0.1, 0, 0, 0, 0
    , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
28             0, 0, 0}; //sets lambda. every 7.5 minutes, step up to next [
    i]//
29     double lambda; //decimal value of average number of customers per uni
    t time//
30 
31     ArrayList<Customer> queue; //long queue to wait for identity check//
```

```
32      ArrayList<Customer> queue1;
33      ArrayList<Customer> queue2;
34      ArrayList<Customer> queue3;
35      ArrayList<Customer> queue4;
36      //4 different queues to go through body scanner//
37      ArrayList<Customer> shortestQueue; //finds shortest queue//
38      int waitingTime; //time it takes to get from the beginning to the end
         of the queue//
39      int numArrivals; //how many people arrive in a minute//
40      int numCustomersServed; //how many people are served throughout the d
        ay?//
41      int totalServed;
42      int averageNumCustomersServed;
43      int[][] timeArray = new int[1050][102]; //holds total queue size at e
        ach minute//
44      int[][] averageArray = new int[1050][102]; //holds average body scann
        er queue size at each minute//
45      int[] waitArray = new int[6000];
46      int[] numCustomersServedArray = new int [100]; //keeps track of the n
        umber of customers served//
47      int available;
48      int available2;
49      int available3;
50      //checks availability of identity check//
51      int availableQueue1;
52      int availableQueue2;
53      int availableQueue3;
54      int availableQueue4;
55      //checks availability of body scanners//
56
57      public Simulate(){
58          //resets and initializes queue//
59          available = 0;
60          available2 = 0;
61          available3 = 0;
62          availableQueue1 = 0;
63          availableQueue2 = 0;
64          availableQueue3 = 0;
65          numArrivals = 0;
66          waitingTime = 0;
67          numCustomersServed = 0;
68          totalServed = 0;
69          queue = new ArrayList<Customer>();
70          queue1 = new ArrayList<Customer>();
71          queue2 = new ArrayList<Customer>();
72          queue3 = new ArrayList<Customer>();
73          queue4 = new ArrayList<Customer>();
74      }
75
```

```
76      private int getArrivals(){
77          double[] poissonArray = new double[16]; //holds the poisson distr
    ibutions//
78
79          Random rand = new Random();
80          double randomDouble = rand.nextDouble();
81
82          poissonArray[0] = poisson(0, lambda);
83          poissonArray[1] = poissonArray[0] + poisson(1, lambda);
84          poissonArray[2] = poissonArray[1] + poisson(2, lambda);
85          poissonArray[3] = poissonArray[2] + poisson(3, lambda);
86          poissonArray[4] = poissonArray[3] + poisson(4, lambda);
87          poissonArray[5] = poissonArray[4] + poisson(5, lambda);
88          poissonArray[6] = poissonArray[5] + poisson(6, lambda);
89          poissonArray[7] = poissonArray[6] + poisson(7, lambda);
90          poissonArray[8] = poissonArray[7] + poisson(8, lambda);
91          poissonArray[9] = poissonArray[8] + poisson(9, lambda);
92          poissonArray[10] = poissonArray[9] + poisson(10, lambda);
93          poissonArray[11] = poissonArray[10] + poisson(11, lambda);
94          poissonArray[12] = poissonArray[11] + poisson(12, lambda);
95          poissonArray[13] = poissonArray[12] + poisson(13, lambda);
96          poissonArray[14] = poissonArray[13] + poisson(14, lambda);
97          poissonArray[15] = poissonArray[15] + poisson(15, lambda);
98          //calls on poisson method to find cumulative distribution//
99
100         if ( randomDouble <= poissonArray[0])
101             return 0;
102         if ( randomDouble <= poissonArray[1] && randomDouble >= poissonAr
    ray[0])
103             return 1;
104         if ( randomDouble <= poissonArray[2] && randomDouble >= poissonAr
    ray[1])
105             return 2;
106         if ( randomDouble <= poissonArray[3] && randomDouble >= poissonAr
    ray[2])
107             return 3;
108         if ( randomDouble <= poissonArray[4] && randomDouble >= poissonAr
    ray[3])
109             return 4;
110         if ( randomDouble <= poissonArray[5] && randomDouble >= poissonAr
    ray[4])
111             return 5;
112         if ( randomDouble <= poissonArray[6] && randomDouble >= poissonAr
    ray[5])
113             return 6;
114         if ( randomDouble <= poissonArray[7] && randomDouble >= poissonAr
    ray[6])
115             return 7;
116         if ( randomDouble <= poissonArray[8] && randomDouble >= poissonAr
```

```
116 ray[7])
117             return 8;
118         if ( randomDouble <= poissonArray[9] && randomDouble >= poissonAr
    ray[8])
119             return 9;
120         if ( randomDouble <= poissonArray[10] && randomDouble >= poissonA
    rray[9])
121             return 10;
122         if ( randomDouble <= poissonArray[11] && randomDouble >= poissonA
    rray[10])
123             return 11;
124         if ( randomDouble <= poissonArray[12] && randomDouble >= poissonA
    rray[11])
125             return 12;
126         if ( randomDouble <= poissonArray[13] && randomDouble >= poissonA
    rray[12])
127             return 13;
128         if ( randomDouble <= poissonArray[14] && randomDouble >= poissonA
    rray[13])
129             return 14;
130         if ( randomDouble <= poissonArray[15] && randomDouble >= poissonA
    rray[14])
131             return 15;
132         return 16;
133         //finds where random double falls in distribution, returns number
     of customers arriving//
134         //the probablity gets exponentially smaller as the number of cust
    omers gets bigger//
135     }
136 
137     public static double poisson(int k, double lambda){
138         double poissn;
139         poissn = 0; //reset//
140 
141         for(int i = 0; i <= k; i++)
142             poissn = (Math.pow(lambda, k) * Math.pow(Math.E, -lambda) / f
    actorial(k));
143         //uses formula to find distribution//
144         return poissn;
145     }
146 
147     public static int factorial(int n) {
148         if (n == 0) {
149             return 1;
150         }
151         else{
152             int fact = 1; // this  will be the result
153             for (int i = 1; i <= n; i++) {
154                 fact *= i;
```

```
155             }
156             return fact;
157         }
158         //finds factorial of k and returns to poisson method//
159     }
160 
161     private  void displayStatistics(){
162         printArray2(timeArray);
163         //prints array of queue sizes//
164 
165     }
166 
167     public void simulate(int column){
168         //runs 100 times, column tells us what number run it is//
169         available = 0;
170         available2 = 0;
171         availableQueue1 = 0;
172         availableQueue2 = 0;
173         availableQueue3 = 0;
174         numArrivals = 0;
175         waitingTime = 0;
176         numCustomersServed = 0;
177         int halfmins = 0;
178         lambda = lambdaArray[0];
179         //resets//
180 
181         for (int time = 0; time < 2100; time++){
182             //for each halfminute//
183             halfmins++; //counts time
184             if (halfmins % 15 == 0){
185                 //if it is 7.5 mins later//
186                 lambda = lambdaArray[halfmins/15]; //step up lambda to ne
    xt [i]//
187             }
188 
189             numArrivals = getArrivals(); //how many customers arrive in h
    alf a minute?//
190             for (int j = 1; j <= numArrivals; j++) {
191                 //place each arrival into the queue//
192                 queue.add(new Customer(time)); //tells customer the arriv
    al time//
193             }
194 
195             //finding shortest queue to add to//
196             shortestQueue = queue1; //sets queue1 as the shortest queue//
197             if(queue2.size() < shortestQueue.size()){
198                 shortestQueue = queue2;
199             }
200             if(queue3.size() < shortestQueue.size()){
```

```
201                 shortestQueue = queue3;
202             }
203             if(queue4.size() < shortestQueue.size()){
204                 shortestQueue = queue4;
205             }
206             //checks if other queues are shorter//
207 
208             if (queue.size() != 0 && available <= time){
209                 //if the queue has people and the server is available, se
    rve a customer//
210                 customer = queue.remove(0); //remove first customer//
211                 available = time + customer.getServiceTime(); //when serv
    er is next available//
212                 shortestQueue.add(customer); //move customer to the short
    est queue//
213             }
214             if (queue.size() != 0 && available2 <= time){
215                 //if the queue has people and the server is available, se
    rve a customer//
216                 customer = queue.remove(0); //remove first customer//
217                 available2 = time + customer.getServiceTime(); //when ser
    ver is next available//
218                 shortestQueue.add(customer); //move customer to the short
    est queue//
219             }
220 
221                 if (queue.size() != 0 && available3 <= time){
222                     //if the queue has people and the server is available
    , serve a customer//
223                     customer = queue.remove(0); //remove first customer//
224                     available3 = time + customer.getServiceTime(); //when
     server is next available//
225                     shortestQueue.add(customer); //move customer to the s
    hortest queue//
226                 }
227 
228             if (queue1.size() != 0 && availableQueue1 <= time){
229                 //if the queue has people and the server is available, se
    rve a customer//
230                 customer = queue1.remove(0); //remove first customer//
231                 availableQueue1 = time + customer.getServiceTime(); //whe
    n server is next available//
232                 numCustomersServed++;
233             }
234             if (queue2.size() != 0 && availableQueue2 <= time){
235                 //if the queue has people and the server is available, se
    rve a customer//
236                 customer = queue2.remove(0); //remove first customer//
237                 availableQueue2 = time + customer.getServiceTime(); //whe
```

```
237 n server is next available//
238                 numCustomersServed++;
239             }
240             if (queue3.size() != 0 && availableQueue3 <= time){
241                 //if the queue has people and the server is available, se
    rve a customer//
242                 customer = queue3.remove(0); //remove first customer//
243                 availableQueue3 = time + customer.getServiceTime(); //whe
    n server is next available//
244                 numCustomersServed++;
245             }
246             if (queue4.size() != 0 && availableQueue3 <= time){
247                 //if the queue has people and the server is available, se
    rve a customer//
248                 customer = queue4.remove(0); //remove first customer//
249                 availableQueue4 = time + customer.getServiceTime(); //whe
    n server is next available//
250                 numCustomersServed++;
251             }
252 
253             if(halfmins % 2 == 0){
254                 averageArray[halfmins/2 - 1][0] = halfmins/2; //saves tim
    es//
255                 int queueAverage = (queue1.size() + queue2.size() + queue
    3.size() + queue4.size()) / 4;
256                 //finds queue size average//
257                 averageArray[halfmins/2 - 1][column] = queueAverage; //sa
    ves queue size average//
258             }
259             if(halfmins % 2 == 0){
260                 timeArray[halfmins/2 - 1][0] = halfmins/2; //saves times/
    /
261                 timeArray[halfmins/2 - 1][column] = queue.size() + averag
    eArray[halfmins/2 - 1][column];
262                 //adds total number of sizes//
263             }
264 
265         }
266 
267         if(column == 100){
268 
269 
270             for(int i = 0; i < 1050; i++){
271                 int sum = 0;
272                 int average = 0;
273                 for(int j = 1; j < 102; j++)
274                     sum = sum + timeArray[i][j];
275 
276 
```

```
277                 average = sum/100;
278                 timeArray[i][101] = average;
279                 //finds average sizes, saves in last column//
280             }
281 
282             displayStatistics();
283             //prints the array at the end//
284         }
285     }
286 
287     private static void printArray2(int[][] anArray){
288         //for(int r=0; r<anArray.length; r++) {
289        //for(int c=0; c<anArray[r].length; c++)
290           //System.out.print(anArray[r][c] + " ");
291        //System.out.println();
292         for(int i = 0;  i < 1050; i++)
293 
294             System.out.println(anArray[i][0] + "\t" + anArray[i][101] );//
/+ "\n");
295         //prints the first and last column of the array//
296 
297             }
298     private static void printArray(int[] anArray) {
299         for (int i = 0; i < anArray.length; i++) {
300             if (i > 0) {
301                 System.out.print(", ");
302             }
303             System.out.print(anArray[i]);
304         }
305         //prints 1d array//
306     }
307 
308     public static void main(){
309         //Scanner scan1 = new Scanner(System.in);
310         //int flightTime;
311         //System.out.println("input flight time in terms of minutes after
 3:00 AM");
312         //flightTime = scan1.nextInt(); //input flight time//
313 
314         int column = 0;
315         Simulate Sim = new Simulate();
316 
317         for(int i = 0; i < 100; i++){
318             column++;
319             Sim.simulate(column); //simulates queue 100 times//
320         }
321 
322     }
323 }
```

```
324
325
326
327
```