```java
/*  Name: Olga Kochepasova
 *  COSC 311 - FA19
 *  pp1008
 *  URL: https://github.com/okochepasova/COSC311/blob/master/p1/Project1.java
 */
import java.util.Random;
import java.util.ArrayList;
import java.util.Collections;


public class Project1 {
    /* Customers' wait time <= 5 minutes each
      The extent of time of interest is 20 minutes
      Each time tick represents 1 minute
      tick time = 100 milliseconds

      At each time tick:
        - Each server that has completed service of a customer "releases" the
        customer.
        - Every customer currently in line has their wait time incremented.
        - A number of new customers enter the system.
        - Each idle server removes one customer from the queue.

      Run the experiment under the following environmental conditions:
        - with 1, 2, 4, 8 servers.
          Serv. Req.      Cust. Arr.
        - 1-12 minutes    2/minutes    heavy demand
        - 1-3 minutes    0.25/minute    light demand
    */


    // Inner Classes
    class Customer {
        int waitTime, serviceTime;

        public Customer(int time) {
            waitTime = 0;
            serviceTime = time;
        }
    }
    class Server {
        private Customer c;

        public Server() {
            c = null;
        }

        public void takeCustomer(Customer c) {
            if (this.isIdle()) this.c = c;
        }
        public void work() {
            if (!this.isIdle()) {
                c.serviceTime--;
                if (c.serviceTime < 1) c = null;
            }
        }
        public boolean isIdle() {
            return c == null;
        }
    }
    class Node {
        public Customer value;
        public Node next;

        public Node(Customer obj) {
            value = obj;
            next = null;
        }
    }
```

```
70      class Queue {
71
72          public Node head, tail;
73          public int length;
74
75          public Queue() {
76              head = tail = null;
77              length = 0;
78          }
79
80          public void add(Customer obj) {
81              Node n = new Node(obj);
82              this.add(n);
83          }
84          public void add(Node n) {
85              if (head == null) head = tail = n;
86              else {
87                  tail.next = n;
88                  tail = n;
89              }
90              length++;
91          }
92          public Node pop() {
93              if (length < 1) return null;
94              //Body
95              Node n = head;
96              head = n.next;
97              if (head == null) tail = null;
98              else n.next = null;
99              // Closing
100             length--;
101             return n;
102         }
103         public boolean isEmpty() {
104             return length<1;
105         }
106     }
107
108
109     // Class Variables
110     private int[] serviceTime;
111     private double arrivalRate;
112     private Queue line;
113     ArrayList<Integer> output;
114     Server[] frontDesk;
115
116
117     // Methods
118     private void populate(int minTime, int maxTime, double rate, int servNum) {
119         arrivalRate = rate;
120         line = new Queue();
121         output = new ArrayList<Integer>();
122
123         serviceTime = new int[2];
124         serviceTime[0] = minTime;
125         serviceTime[1] = maxTime;
126
127         frontDesk = new Server[servNum];
128         for (int i=0; i<servNum; i++) {
129             frontDesk[i] = new Server();
130         }
131     }
132
133     private int getCustomerTime() {
134         if (serviceTime == null) return -1;
135         return getRandomNumberInRange(serviceTime[0],serviceTime[1]);
136     }
137
138     public void run() throws InterruptedException {
```

```
139          long end = System.currentTimeMillis() + 2000; //20 ticks
140          int tick = 0;
141          Node n; //Customer line Node
142          while(System.currentTimeMillis() < end) {
143              // customers come in
144              int x = getPoissonRandom(arrivalRate);
145              for(int i=0; i<x; i++) {
146                  Customer c = new Customer( getCustomerTime() );
147                  line.add(c);
148              }
149
150              // idle servers take customers
151              for (int i = 0; i < frontDesk.length && !line.isEmpty(); i++) {
152                  if (frontDesk[i].isIdle()) {
153                      n = line.pop();
154                      output.add(n.value.waitTime);
155                      frontDesk[i].takeCustomer(n.value);
156                  }
157              }
158
159              // servers process customers
160              for (Server desk:frontDesk) {
161                  if (!desk.isIdle()) desk.work();
162              }
163
164              // customers' waitTime increments
165              for (n = line.head; n != null; n = n.next) {
166                  n.value.waitTime++;
167              }
168
169              // Tick Output
170              System.out.println("Tick#: "+tick+"\n"+this);
171              tick++;
172              Thread.sleep(100); //1 tick
173          }
174      }
175
176      private int buisyServers() {
177          int num = 0;
178          for (Server r:frontDesk) {
179              if (!r.isIdle()) num++;
180          }
181          return num;
182      }
183      private int totalWaitTime() {
184          int sum = 0;
185          for (int i:output) sum += i;
186          for (int i:queueWaitTime()) sum += i;
187          return sum;
188      }
189      private ArrayList<Integer> queueWaitTime() {
190          ArrayList<Integer> list = new ArrayList<Integer>();
191          for (Node n=line.head;n!=null;n=n.next) list.add(n.value.waitTime);
192          return list;
193      }
194
195      public String toString() {
196          ArrayList<String> l = new ArrayList<String>();
197          String s = "\t%s: %d\n";
198
199          int i = buisyServers();
200          l.add(String.format(s,"# Customers in service",i));
201          l.add(String.format(s,"# Customers with completed service",
202  output.size()-i));
203          l.add(String.format(s,"# Customers in queue",line.length));
204
205          i = totalWaitTime();
206          l.add(String.format(s,"Total wait time", i));
207
```

```
208          ArrayList<Integer> list = queueWaitTime();
209          list.addAll(output);
210          double ave = Double.valueOf(i)/(list.size());
211          l.add(String.format("\tWait time: %d, %f, %d\n",Collections.min(list),
212    ave,Collections.max(list)));
213
214          return String.join("", l);
215      }
216
217
218      // Static Methods
219      private static int getPoissonRandom(double mean) {
220          Random r = new Random();
221          double L = Math.exp(-mean);
222          int k = 0;
223          double p = 1.0;
224          do {
225              p = p * r.nextDouble();
226              k++;
227          } while (p > L);
228          return k - 1;
229      }
230
231      /* START CODE BLOCK
232        Got the code from:
233        https://www.mkyong.com/java/java-generate-random-integers-in-a-range/
234      */
235      private static int getRandomNumberInRange(int min, int max) {
236          if (min >= max) {
237              throw new IllegalArgumentException("max must be greater than min");
238          }
239
240          Random r = new Random();
241          return r.nextInt((max - min) + 1) + min;
242      }
243      // END CODE BLOCK
244
245
246      /*
247        Provide output to confirm your code is correct in the case where service
248        requirement is 1-3 minutes, with 2 arrivals/minute, and 4 servers.
249      */
250      public static void main(String[] args) {
251          Project1 shop = new Project1();
252          try {
253              shop.populate(1,3,2.0,4);
254              shop.run();
255          } catch(InterruptedException e) {
256              System.out.println("Something went wrong.");
257          }
258      }
259  }
260
```