Lab №07

Alyssandra Cordero, Mikhael Opeyemi, Emanuel Robles 12/05/2019

Listing 1: Lab 07 -Descrete Event Simulator

```
1  /**************************************************************************
2   * The Node class contains the necessary constructors to create a Node.
3   *************************************************************************/
4  public class Node{
5   //Variables declaration.
6   String data;
7   Node next;
8   //Constructor 1.
9   Node(String data){
10    this.data = data;
11    next = null;
12  }
13  //Constructor 2.
14  Node(String data, Node next){
15    this.data = data;
16    this.next = next;
17  }
18  /**********************************
19   *Prints the data inside the node.
20   *@return The data inside the node.
21   *********************************/
22  public String toString(){
23    return data;
24  }
25 }
26 import java.io.*;
27 import java.util.*;
28 /*****************************************************************************
29  * The UberUser class simulates an Uber user.
30  ****************************************************************************/
31 public class UberUser implements Comparable<UberUser>{
32  // fields(this)
33  private String name;
34  private String pickUp;
35  private String destination;
36  private int rideTime;
37  private int miles;
38  private double earning;
```

```
39  /*****************************************************************************
40    * This constructor is used as a default constructor and sets default ↩
          values.
41    *****************************************************************************/
42  public UberUser() {
43    this.name = "";
44    this.pickUp = "";
45    this.destination = "";
46    this.rideTime = 0;
47    this.miles = 0;
48    this.earning = 0.0;
49  }
50  /*****************************************************************************
51    *This constructor is used as a reference for the testers to create ↩
          multiple objects and it links
52    *the values given from the file to the objects.
53    *@param s holds the information of the log file.
54    *****************************************************************************/
55  public UberUser(String s) {
56    String [] tokens = s.split(", ");
57    this.name = tokens[0];
58    this.pickUp = tokens[1];
59    this.destination = tokens[2];
60    this.rideTime = Integer.parseInt(tokens[3]);
61    this.miles = Integer.parseInt(tokens[4]);
62    this.earning = Double.parseDouble(tokens[5]);
63  }
64  //getters
65  public String getName() {
66    return this.name;
67  }
68  public String getPickUp() {
69    return this.pickUp;
70  }
71  public String getDest() {
72    return this.destination;
73  }
74  public int getRideTime() {
75    return this.rideTime;
76  }
77  public int getMiles() {
78    return this.miles;
79  }
80  public double getEarning() {
81    return this.earning;
82  }
83
```

```
84   //setters
85   public void setName(String n) {
86     this.name = n;
87   }
88   public void setPickUp(String p) {
89     this.pickUp = p;
90   }
91   public void setDest(String d) {
92     this.destination = d;
93   }
94   public void setRideTime(int r) {
95     this.rideTime = r;
96   }
97   public void setMiles(int m) {
98     this.miles = m;
99   }
100  public void setEarning(double e) {
101    this.earning = e;
102  }
103  /*****************************************************************************
104    Checks if two rides are the same.
105    @param o The object to compare to.
106    @return true If the two rides are the same.
107    @return false If the rides are not the same.
108    *****************************************************************************/
109  public boolean equals(UberUser o) {
110    if (this.name == o.getName() && this.pickUp == o.getPickUp() ↩
         &&this.destination == o.getDest() &&
111        this.rideTime == o.getRideTime() && this.miles == o.getMiles() && ↩
             this.earning == o.getEarning())
112      return true;
113    else
114      return false;
115  }
116  /*****************************************************************************
117    Compares the files of two objects to prioritize rides for the driver.
118    @param o The object to compare with.
119    @return 1 If the current object should be prioritized.
120    @return -1 If the object we are comparing to should be prioritized.
121    @return 0 In case the information is the same.
122    *****************************************************************************/
123  public int compareTo(UberUser o) {
124    if (this.rideTime > o.getRideTime())
125      return 1;
126    else if (this.rideTime < o.getRideTime())
127      return -1;
128    else {
```

```
129       if (this.miles > o.getMiles())
130         return 1;
131       else if (this.miles < o.getMiles())
132         return -1;
133       else {
134         if (this.earning > o.getEarning())
135           return 1;
136         else if (this.earning < o.getEarning())
137           return -1;
138         else
139           return 0;
140       }
141     }
142   }
143   /*****************************************************************************
144     Prints the information of each field.
145     *****************************************************************************/
146   public String toString() {
147     return "Pickup up: "+this.name+" from: "+this.pickUp+" to: ↩
          "+this.destination+". Total time: "+
148       this.rideTime+". Total miles: "+this.miles+". You will earn: ↩
          $"+this.earning+".";
149   }
150   /*****************************************************************************
151     The MyLinkedList class is used to create a linked list.
152     *****************************************************************************/
153   public static class MyLinkedList{
154     //Nodes initialization.
155     Node first, last;
156     //Constructor
157     public MyLinkedList(){
158       first = last = null;
159     }
160     /************************************
161       * Checks if the Linked list is empty.
162       * @return true if it is empty.
163       ************************************/
164     public boolean isEmpty(){
165       return first == null;
166     }
167     /************************************
168       * Empties the Linked list.
169       ************************************/
170     public void makeEmpty(){
171       first = last = null;
172     }
173     /************************************
```

```
174       * Adds an element to the Linked list.
175       ************************************/
176     public void add(String s){
177       Node n = new Node(s);
178       // in case is empty
179       if(isEmpty()){
180         first = last = n;
181       }
182       else{//otherwise
183         last.next = n;
184         last = n;
185       }
186     }
187     /************************************
188       * Returns the size of the Linked list.
189       * @return Size of the Linked list.
190       ************************************/
191     public int size(){
192       int total = 0;
193       Node d = first;
194       while(d != null){
195         d = d.next;
196         total++;
197       }
198       return total;
199     }
200     /************************************************
201       * Checks if an element is in the Linked list.
202       * @return true if the item its found.
203       ************************************************/
204     public boolean find(String t) {
205       Node d = first;
206       while (d != null) {
207         if (d.data == t)
208           return true;
209         d = d.next;
210       }
211       return false;
212     }
213   }
214 }
215 import java.io.*;
216 import java.util.*;
217 /*****************************************************************************
218   * Implements a simulator for UberUser class.
219   *****************************************************************************/
220 public class Simulator extends UberUser{
```

```
221   //Variables initialization.
222   private static int clock = 0;              //Used to keep track of the ↩
          clock.
223   private static int END;                    //Used to know when to stop ↩
          the clock.
224
225   private static PriorityQueue<UberUser> q;  //Priority queue. Used to ↩
          store pending trips.
226   private static boolean inTransit = true;   //The value is true while ↩
          the uber user is driving.
227   private static UberUser[] rides;           //rides array initialization.
228
229   public static Scanner input = new Scanner(System.in); //Stores user input.
230
231   //Counters for the simulation
232   private static int numberOfRequests;
233   private static int numberOfRequestsNot;
234   private static long waitingTime;
235   private static long avgWaitingTime;
236   private static long timeOfSimulation; //Stores how much time did the ↩
          simulation ran.
237
238   //Variables to calculate time
239   private static long start;
240   private static long end;
241
242   public static void main (String [] args) throws Exception{
243
244     MyLinkedList holder = new MyLinkedList();
245     Scanner info = new Scanner(new File("uberLog.txt"));
246     while (info.hasNext()) {
247       String inf = info.nextLine();
248       holder.add(inf);
249     }
250     rides = new UberUser[holder.size()];// to keep track of all the rides
251     q = new PriorityQueue<UberUser>();// to process the priority of the ↩
          rides
252
253     //Creates the uber user objects.
254     Node d = holder.first;
255     for (int i = 0; i < rides.length; i++) {
256       rides[i] = new UberUser(d.data);
257       q.add(rides[i]);
258       d = d.next;
259     }
260     //Asking the user the desired total driving time
261     System.out.println("How much time would you like to drive?");
```

```
262     END = input.nextInt();
263     int countM = 0; //this is the counter for the total number of miles ↩
            driven
264     //the simulation
265
266     while(inTransit){
267
268       start = System.nanoTime();//Start count.
269       UberUser dum = q.poll();
270       end = System.nanoTime();//end count.
271
272       timeOfSimulation = end - start;
273
274       numberOfRequests++; //Counts the number of rides.
275
276       Thread.sleep(3000);
277       clock += dum.getRideTime();
278       countM += dum.getMiles();
279       System.out.println("next in line:"+dum);
280       if(clock >= END || q.isEmpty())
281         inTransit = false;
282
283     }
284     numberOfRequestsNot = (rides.length - numberOfRequests);
285
286     // reprot in the statistics file
287     PrintWriter stats = new PrintWriter(new File("statistics.txt"));
288     stats.println("Uber Driver's total ride time is "+clock + " minutes.");
289     stats.println("Uber Driver drove for "+countM+" miles.");
290     stats.println("The total number of rides processed is "+ ↩
            numberOfRequests+".");
291     stats.println("The total number of rides that stayed in the queue is ↩
            "+ numberOfRequestsNot+".");
292     stats.println("The time of simulation is "+ timeOfSimulation +" ↩
            nanoseconds.");
293     stats.close();
294
295   }
296 }
```