```
1
  /*================================================================================
  ================================
2 * @title Payless Medical Service
3 *   Event Based Simulation
4 * @author Bobby Purcell
5 * @description : not exactly my magnum opus but it works well enough
6 * Events:
7 *   1- PatientArrive, calls
8 *   2- PatientDeath
9 *   3- PatientTreatment, removes death event
10 *   4- SimulationEnd, calcs stats
11 *
12
  ================================================================================
  ================================*/
13 package CSC318.EventBased;
14
15 import java.util.ArrayList;
16
17 @SuppressWarnings("unchecked")
18 public class EventBasedClinicSimulation {
19     static double totalTimeWaitHeart = 0.0, totalTimeWaitGastro = 0.0,
    totalTimeWaitBleed = 0.0;
20
21     public static void main(String[] args) {
22
23         double bigTime = 0.0; //the master clock
24         double timeToRun = 6000; //6000 minutes = 100hrs
25         double eventTime;// the event time
26         double deltime; //change in time
27         //TODO: setup patient death time tracking something something
28
29         GenericManager eventQ = new GenericManager<>(); //order of
  events
30         GenericManager patQ = new GenericManager<>(); //patients in
  waiting room
31         int patientID = 0;  //unique id for patients (and their death
  event when appropriate)
32         double numDocs = 1.0;     //how many docs are treating patients
    at the clinic
33         int numWaiting;
34         int numEvent;
35         int totalHeart = 0, totalGastro = 0, totalBleed = 0,
36                 totalHeartDead = 0, totalGastroDead = 0,
  totalBleedDead = 0;
37
38         //total wait time for each patient type (for avgs)
```

```
39
40
41          //Makes new patient Patient(number,ailment)
42          //adds the new patient to the arraylist of events in order
43          //prime the Queue
44          patientID++;
45          eventQ.addFront(new Event(0, 1, patientID));
46          eventQ.addEnd(new Event(timeToRun, 4, -9999));
47          Event current = (Event) eventQ.getValue(0);
48          while (current.getEventType() != 4) {
49              deltime = current.getTime() - bigTime;
50
51              bigTime = current.getTime();
52              switch (current.eventType) {
53                  case 1: // arrival event
54                      //new patient
55                      Patient p = new Patient(patientID, bigTime);
56                      patQ.addInOrder(p);
57                      //gen new treatmentevent
58                      eventTime = (bigTime + TimeToTreat(p.
   getAilmentType(), numDocs));
59                      Event e = new Event(eventTime, 3, patientID);
60                      //takes sum of all patient treatment time ahead of
    current
61                      eventQ.addInOrder(e);
62                      //gen new arrival
63                      Event ae = new Event((bigTime + TimeToArrive()),
   1, patientID++);
64                      eventQ.addInOrder(ae);
65                      //gen new death
66                      Event de = new Event((bigTime + p.gettDeath()), 1
   , patientID++);
67                      eventQ.addInOrder(ae);
68                      break;
69                  case 2: // death event
70                      //patient died before treatment, remove from Qs
71                      //resolve and track the dead patient
72                      int died = KillPatient(current.getPatient(), patQ)
   ;
73                      switch (died) {
74                          case 1:
75                              totalHeartDead += 1;
76                              break;
77                          case 2:
78                              totalGastroDead += 1;
79                              break;
80                          case 3:
81                              totalBleedDead += 1;
```

```
82                              break;
83                          default:
84                              System.err.printf("Tried to kill patient
    %d, didnt find them.", current.getPatient());
85                      }
86                      //remove treatment event
87                      KillPatientEvent(current.getPatient(), 3, eventQ)
   ;
88
89                      break;
90                  case 3: // treatment event
91                      //patient treated before death
92                      //resolve and track the patient
93                      int treated = TreatPatient(current.getPatient(),
   patQ);
94                      switch (treated) {
95                          case 1:
96                              totalHeart += 1;
97                              break;
98                          case 2:
99                              totalGastro += 1;
100                             break;
101                         case 3:
102                             totalBleed += 1;
103                             break;
104                         default:
105                             System.err.printf("Tried to treat patient
    %d, didnt find them.", current.getPatient());
106                     }
107                     //remove death event
108                     KillPatientEvent(current.getPatient(), 2, eventQ)
   ;
109
110                     break;
111                 case 4: // end simulation event
112                     //go home end sim event youre drunk.
113                     System.err.println("should not be here and if i
   am i've got event proc issues");
114                     System.exit(1);
115                     break;
116             }
117         //  eventQ.sort();
118         //  patQ.sort();
119
120           //cycle to next event
121           eventQ.managedRemove(0);
122           current = (Event) eventQ.getValue(0);
123
```

```
124         }//end of while(not event 4)
125         //todo: Its reportin time
126         int totalTreated = totalBleed+totalGastro+totalHeart;
127         int totalDead = totalBleedDead+totalGastroDead+totalHeartDead
    ;
128         System.out.println("Total Treated =" + totalTreated);
129         System.out.println("Total Dead =" + totalDead);
130     }
131 
132     private static int KillPatient(int patient, GenericManager patQ)
    {
133         boolean removedp = false;
134         Patient p = null;
135         //search the ques for matching items and remove them
136         for (int i = 0; i < patQ.getCount(); i++) {
137             p = (Patient) patQ.getValue(i);
138             if (p.getID() == patient) {
139                 patQ.managedRemove(i);
140                 removedp = true;
141             }
142         }
143 
144         if (removedp) return p.getAilmentType(); //returns what
    patient ailment caused death
145         else return -1;
146     }
147 
148     private static int TreatPatient(int patient, GenericManager patQ)
     {
149         boolean removedp = false;
150         Patient p = null;
151         //search the ques for matching items and remove them
152         for (int i = 0; i < patQ.getCount(); i++) {
153             p = (Patient) patQ.getValue(i);
154             if (p.getID() == patient) {
155                 patQ.managedRemove(i);
156                 removedp = true;
157             }
158         }
159 
160         if (removedp) return p.getAilmentType(); //returns what
    patient ailment was treated
161         else return -1;
162     }
163 
164     private static int KillPatientEvent(int patient, int eventType,
    GenericManager eventQ) {
165         boolean removede = false;
```

```
166         Event e = null;
167         //search the ques for matching items and remove them
168         for (int i = 0; i < eventQ.getCount(); i++) {
169             e = (Event) eventQ.getValue(i);
170             if (e.getPatient() == patient && e.getEventType() ==
    eventType) {
171                 eventQ.managedRemove(i);
172                 removede = true;
173             }
174         }
175         if (removede) return e.eventType; //returns what event was
    removed
176         else return -1;
177     }
178 
179 
180     //generates new patient arrival from rate 3/hr
181     public static double TimeToArrive() {
182         double deltime;
183         double bigx = Math.random();
184         if (bigx > .9) bigx = Math.random();
185         deltime = -Math.log(1.0 - bigx) / 3.0;
186         return deltime;
187     }//end timetoarrive
188 
189     //generates new patient arrival from rate 3/hr
190     public static double TimeToTreat(int a, double numDocs) {
191         double timeTreat;
192         double bigx = Math.random();
193         double rate = 0.0; //number of patients/hr
194 
195         switch (a) {
196 
197             case 1://Heart
198                 rate = 2.0;
199                 break;
200             case 2://Gastro
201                 rate = 4.0;
202                 break;
203             case 3://Bleed
204                 rate = 6.0;
205                 break;
206             default:
207                 System.err.println("Wtf? This patient doesnt have an
    illness! Literally impossible.");
208                 System.exit(1);//there is a serious problem, exit
209         }
210         timeTreat = 60 * Math.log(1.0 - bigx) / -(rate * numDocs);
```

```
211         return timeTreat;
212     }//end timetoarrive
213 
214 
215 }//end EventBasedClinicSimulation
216 
217 
    /*=================================================================
    ================================
218 Patient class
219 represents a patient at the clinic
220 Has Patient ID, Type of ailment, and arrival, wait, and total time
221 
    =================================================================
    ================================*/
222 class Patient implements Comparable {
223     protected int ailmentType;  //1= heart,2=gastro, 3=bleeding
224     //arrival time, time waited, time till death, time in system
225     protected double tArrive;
226     protected double tWait;
227     protected double tDeath;
228     protected int myDeath;
229     protected int ID; //patient ID
230 
231     public Patient(int ID, double bigTime) {
232         this.ID = ID;
233         this.myDeath = ID;
234         this.ailmentType = setAilmentType();
235         settDeath(); //generate patient death time
236         tArrive = bigTime;
237     }
238 
239     public int getID() {
240         return ID;
241     }
242 
243     public int getMyDeath() {
244         return myDeath;
245     }
246 
247     public int getAilmentType() {
248         return ailmentType;
249     }
250 
251     private int setAilmentType() {
252         int r;
253 
254         int x = ((int) (Math.random() * 100));
```

```
255         if (x < 30) {
256             r = 1; //Heart
257         } else if (x < 50) {
258             r = 2;//Gastro
259         } else r = 3;//Bleed
260         return r;
261     }
262 
263 
264     public double gettArrive() {
265         return tArrive;
266     }
267 
268     public void settArrive(double tArrive) {
269         this.tArrive = tArrive;
270     }
271 
272     public double gettWait() {
273         return tWait;
274     }
275 
276     public void settWait(double tWait) {
277         this.tWait += tWait;
278     }
279 
280     public double gettDeath() {
281         return tDeath;
282     }
283 
284     private void settDeath() {
285         double dTimer;
286         int t = getAilmentType();
287         double mu = 0.0, sigma = 0.0;
288         switch (t) {
289 
290             case 1:
291                 mu = 10;
292                 sigma = 35;
293                 break;
294             case 2:
295                 mu = 30;
296                 sigma = 80;
297                 break;
298             case 3:
299                 mu = 20;
300                 sigma = 65;
301                 break;
302             default:
```

```
303                 System.err.println("Wtf? This patient doesnt have an illness and expects to die!");
304                 System.exit(1);//there is a serious problem, exit
305         }
306 
307         dTimer = sigma * (Math.random()) + mu;
308         tDeath = dTimer;
309     }
310 
311     @Override
312     public int compareTo(Object o) {
313         if (gettWait() > ((Patient) o).gettWait()) {
314             return 1;
315         } else if (gettWait() < ((Patient) o).gettWait()) {
316             return -1;
317         } else {
318             return 0;
319         }
320 
321     }
322 }
323 
324 
    /*=====================================================================
    ==================================
325 GenericManager class
326 represents an array list of objects that can be compared to each other
327 objects can be added first, last, or added in order by comparator
328 mostly a copy of Kent Pickett's code - If it ain't broke don't fix it.
329 
    =====================================================================
    ================================*/
330 class GenericManager<T extends Comparable<? super T>> {
331 
332     protected ArrayList<T> list = new ArrayList<>();
333     protected int count; //items in the arraylist
334 
335     //generic constructor
336     public GenericManager() {
337         //initialize to 0
338         this.count = 0;
339     }
340 
341     public int getCount() {
342         return count;
343     }
```

```
344
345     public int addFront(T x) {
346         list.add(0, x);
347         count++;
348         return count;
349     }
350
351     public int addEnd(T x) {
352         list.add(count++, x);
353         return count;
354     }
355
356     //adds object x to the list a the position determined by its
    comparator
357     public int addInOrder(T x) {
358         int i;
359         if ((count == 0)
360                 || ((x.compareTo(list.get(0)) == -1)
361                 || x.compareTo(list.get(0)) == 0)) {
362             //object goes at the front of the list
363             list.add(0, x);
364         } else if ((x.compareTo(list.get(count - 1)) == 1)
365                 || (x.compareTo(list.get(count - 1)) == 0)) {
366             //object goes at the end of the list
367             list.add(count, x);
368         } else {
369             //object goes somewhere in the middle of the list
370             i = 0;
371             //compare x with the list from start until x > the
    current item
372             while ((i < count) && (x.compareTo(list.get(i)) == 1)) i
    ++;
373             //add x in its place after the current item
374             list.add(i, x);
375         }
376         count++;
377         return count;
378     }
379
380     public T getValue(int i) {
381         if (i < count) {
382             return list.get(i);
383         } else {
384             System.err.println(String.format("Attempted to get value
    from a position that doesn't exist: %d", i));
385             return list.get(0);//default case
386         }
387     }
```

```
388
389     //basic generic sorting method, uses object's compareTo
390     public void sort() {
391         T xsave, ysave, a, b;
392         int isw = 1; //is the list sorting
393         int xlast = list.size();
394         while (isw == 1) {
395             isw = 0;
396             for (int i = 0; i <= xlast - 2; i++) {
397                 a = list.get(i);
398                 b = list.get(i + 1);
399                 switch (a.compareTo(b)) {
400                     case 1://already sorted, break
401                         break;
402                     case -1://objects out of order, sort
403                         xsave = list.get(i);
404                         ysave = list.get(i + 1);
405                         list.remove(i);
406                         list.add(i, ysave);
407                         list.remove(i + 1);
408                         list.add(i + 1, xsave);
409                         isw = 1;
410                         break;
411                     default://objects assumed to be equal
412                 }
413             }
414         }
415     }//end of sorting method
416
417     //removes item at specified index and decrements the count
418     public void managedRemove(int i) {
419         if ((i >= 0) && (i <= count - 1)) {
420             list.remove(i);
421             count--;
422         }
423     }
424
425 }//end of generic mgr
426
427
    /*=================================================================
    ================================
428 Event class
429 represents event type, when the event occurs, and which patient it
    belongs to
430 modified from Kent Pickett's code
431
    =================================================================
```

```
431 ================================*/
432 class Event implements Comparable {
433 
434     protected int eventType; // event type
435     protected int patient; // which patient this event belongs to
436     private double time; //when this even occurs
437 
438     public Event(double time, int eventType, int patient) {
439         this.eventType = eventType;
440         this.time = time;
441         this.patient = patient;
442     }
443 
444     @Override
445     //compares based on the events' times
446     public int compareTo(Object o) {
447         if (getTime() > ((Event) o).getTime()) {
448             return 1;
449         } else if (getTime() < ((Event) o).getTime()) {
450             return -1;
451         } else {
452             return 0;
453         }
454 
455     }
456 
457     public int getPatient() {
458         return patient;
459     }
460 
461     public int getEventType() {
462         return eventType;
463     }
464 
465     public double getTime() {
466         return time;
467     }
468 }//end of Event
469 
```