```
1
  /*================================================================
  ================================
2 * @title Payless Medical Service
3 *   Event Based Simulation
4 * @author Bobby Purcell
5 * @description :
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 //enumerated representation of events
18 enum EventType {
19     ARRIVAL, DEATH, TREATMENT, END
20 }
21
22 @SuppressWarnings("unchecked")
23 public class EventBasedClinicSimulation {
24     public static void main(String[] args) {
25
26         double bigTime = 0.0; //the master clock
27         double timeToRun = 6000; //6000 minutes = 100hrs
28         double eventTime;// the event time
29
30         GenericManager eventQ = new GenericManager<>(); //order of
  events
31         GenericManager patQ = new GenericManager<>(); //patients in
  waiting room
32         int patientID = 1;  //unique id for patients (and their death
  event when appropriate)
33         double numDocs = 1.0;    //how many docs are treating patients
   at the clinic
34         double currentWait;
35         int numEvent = 0;
36         int totalHeart = 0, totalGastro = 0, totalBleed = 0,
37                 totalHeartDead = 0, totalGastroDead = 0,
  totalBleedDead = 0;
38
39         //total wait time for each patient type (for avgs)
```

```
40
41
42          //Makes new patient Patient(number,ailment)
43          //adds the new patient to the arraylist of events in order
44          //prime the Queue
45          eventQ.addFront(new Event(0, EventType.ARRIVAL, patientID));
46          eventQ.addEnd(new Event(timeToRun, EventType.END, -9999));
47          Event current;
48
49          while (bigTime <= timeToRun) {
50              numEvent++;
51              current = (Event) eventQ.getValue(0);
52
53              bigTime = current.getTime();
54              switch (current.eventType) {
55                  case ARRIVAL: // arrival event
    ------------------------------------------------------------------------
    --
56
57                      //new patient
58                      Patient p = new Patient(patientID, current.getTime
    ());
59
60                      //add patient to line
61                      patQ.addInOrder(p);
62
63                      //how long is the current waiting lines total
    treatment time? (sum of treatment times)
64                      currentWait = CalcCurrentWait(patQ);
65
66                      //how long is p's treatment time?
67                      double pTreatTime = TimeToTreat(p.getAilmentType()
    , numDocs);
68
69                      //set this patients treatment time to how long it
    will take to treat them
70                      p.settTreat(pTreatTime);
71
72                      //set this patients wait time to how long the
    current wait is counting their treatment time
73                      p.settWait(currentWait);
74
75                      //gen new treatmentevent
76                      // (current big time, plus ps treatment time, plus
     the people ahead of p in line)
77                      eventTime = (p.gettWait());
78                      Event te = new Event(eventTime, EventType.
    TREATMENT, patientID);
```

```
 79                      eventQ.addInOrder(te);
 80
 81                      System.out.println("Patient " + patientID + "
    treatment event created at event time\t" + eventTime
 82                              + "\n\t\t\t\t\t\t\t\t from bigTime " +
    bigTime);
 83
 84                      //gen new death event
 85                      eventTime = (bigTime + p.gettDeath());
 86                      Event de = new Event(eventTime, EventType.DEATH,
    patientID);
 87                      eventQ.addInOrder(de);
 88
 89                      patientID++;
 90                      //gen new arrival
 91                      try {
 92                          eventTime = (bigTime + TimeToArrive());
 93                      } catch (Exception e) {
 94                          e.printStackTrace();
 95                          System.exit(1);
 96                      }
 97                      Event ae = new Event(eventTime, EventType.ARRIVAL
    , patientID);
 98                      eventQ.addInOrder(ae);
 99 //                       System.out.println("Patient " + patientID + "
    arrival event created at event time\t" + eventTime
100 //                               + "\n\t\t\t\t\t\t\tfrom bigTime " +
    bigTime);
101
102                     break;
103                 case DEATH: // death event
    ------------------------------------------------------------------
    ---
104                     //patient died before treatment, remove from Qs
105                     //resolve and track the dead patient
106                     int died = KillPatient(current.getPatient(), patQ
    );
107                     switch (died) {
108                         case 1:
109                             totalHeartDead += 1;
110                             break;
111                         case 2:
112                             totalGastroDead += 1;
113                             break;
114                         case 3:
115                             totalBleedDead += 1;
116                             break;
117                         case -1:
```

```
118                             System.err.printf("Tried to kill patient
     %d, didnt find them.\n", current.getPatient());
119                         }
120                         //remove treatment event
121                         RemovePatientEvent(current.getPatient(), bigTime,
     EventType.TREATMENT, eventQ);
122 
123                         break;
124                     case TREATMENT: // treatment event
    ------------------------------------------------------------------
    ---
125                         //patient treated before death
126                         //resolve and track the patient
127                         int treated = TreatPatient(current.getPatient(),
    patQ);
128                         //decrement everyone elses wait time by this
    death/treatment
129 
130                         switch (treated) {
131                             case 1:
132                                 totalHeart += 1;
133                                 break;
134                             case 2:
135                                 totalGastro += 1;
136                                 break;
137                             case 3:
138                                 totalBleed += 1;
139                                 break;
140                             case -1:
141                                 System.err.printf("Tried to treat patient
     %d, didnt find them.\n", current.getPatient());
142                         }
143                         //remove death event
144                         RemovePatientEvent(current.getPatient(), bigTime,
     EventType.DEATH, eventQ);
145 
146                         break;
147                     case END: // end simulation event
    ----------------------------------------------------------
148 
149                         //todo: Its reportin time
150                         int totalTreated = totalBleed + totalGastro +
    totalHeart;
151                         int totalDead = totalBleedDead + totalGastroDead
    + totalHeartDead;
152                         int numpats = ((Patient) patQ.getValue(0)).getID
    () - 1;
153 
```

```
154
155                     System.out.printf("Total Patients = %d\n\n",
    numpats);
156
157                     System.out.printf("Total Treated = %d  Bleed: %d
        Gas: %d     Heart: %d   \n",
158                                 totalTreated, totalBleed, totalGastro,
    totalHeart);
159
160                     System.out.printf("Total Dead = %d  Bleed: %d
    Gas: %d     Heart: %d   \n",
161                                 totalDead, totalBleedDead,
    totalGastroDead, totalHeartDead);
162
163                     System.out.println("Last Patient ID = " +
    patientID);
164                     System.out.println("Total Events = " + numEvent);
165
166                     System.exit(0);
167             }
168
169             //cycle to next event
170             eventQ.managedRemove(0);
171             current = (Event) eventQ.getValue(0);
172
173         }//end of while(not event 4)
174
175         //todo: Its reportin time
176         int totalTreated = totalBleed + totalGastro + totalHeart;
177         int totalDead = totalBleedDead + totalGastroDead +
    totalHeartDead;
178         int numpats = ((Patient) patQ.getValue(0)).getID() - 1;
179
180
181         System.out.printf("Total Patients = %d\n\n", numpats);
182
183         System.out.printf("Total Treated = %d  Bleed: %d    Gas: %d
      Heart: %d   \n",
184                 totalTreated, totalBleed, totalGastro, totalHeart);
185
186         System.out.printf("Total Dead = %d  Bleed: %d    Gas: %d
    Heart: %d   \n",
187                 totalDead, totalBleedDead, totalGastroDead,
    totalHeartDead);
188
189         System.out.println("Last Patient ID = " + patientID);
190         System.out.println("Total Events = " + numEvent);
191
```

```
192             System.exit(0);
193 
194         }
195 
196         private static double CalcCurrentWait(GenericManager pQ) {
197             double currentWait = 0.0;
198             //get the sum of the treatment time for each person in line
199             for (int i = 0; i < pQ.count; i++) {
200                 currentWait += ((Patient) (pQ.getValue(i))).gettTreat();
201             }
202             return currentWait;
203         }
204 
205         private static int KillPatient(int patient, GenericManager patQ)
        {
206             boolean removedp = false;
207             Patient tp;
208             Patient p;
209             //removes treatment
210             //search the ques for matching items and remove them
211             for (int i = 0; i < patQ.getCount(); i++) {
212                 p = (Patient) patQ.getValue(i);
213                 if (p.getID() == patient) {
214                     //decrements the wait time for everyone in line
215                     for (int j = 0; j < patQ.getCount(); j++) {
216                         tp = ((Patient) patQ.getValue(j));
217                         tp.settWait(tp.gettWait() - p.gettTreat());
218                     }
219                     patQ.managedRemove(i);
220                     patQ.sort();
221                     return p.getAilmentType();
222                 }
223             }
224 
225             return -1;
226         }
227 
228         private static int TreatPatient(int patient, GenericManager patQ)
         {
229             Patient patientToRem;
230             Patient tp;
231             //removes death
232             //search the ques for matching items and remove them
233             for (int i = 0; i < patQ.getCount(); i++) {
234                 patientToRem = (Patient) patQ.getValue(i);
235                 if (patientToRem.getID() == patient) {
236                     //decrements the wait time for everyone in line
237                     for (int j = 0; j < patQ.getCount(); j++) {
```

```
238                     tp = ((Patient) patQ.getValue(j));
239                     tp.settWait(tp.gettWait() - patientToRem.
    gettTreat());
240                 }
241                 patQ.managedRemove(i);
242                 patQ.sort();
243 
244                 return patientToRem.getAilmentType(); //returns what
    patient ailment was treated
245             }
246         }
247         return -1;
248     }
249 
250     private static EventType RemovePatientEvent(int patient, double
    bigTime, EventType eventType, GenericManager eventQ) {
251 
252         Event e;
253         Event e2;
254         //search the ques for matching items and remove them
255         for (int i = 0; i < eventQ.getCount(); i++) {
256             e = (Event) eventQ.getValue(i);
257 
258             if (e.getPatient() == patient && e.getEventType() ==
    eventType) {
259 
260 //                  //decrements the time for all other events for
    TREATMENTs that are removed infront of them
261 //                  //since the treatment is removed, move other
    treatment events up
262 //                  if (eventType == EventType.TREATMENT) {
263 //                      for (int j = 0; j < eventQ.getCount(); j++) {
264 //                          e2 = ((Event) eventQ.getValue(j));
265 //                          //event e is being removed from the Q, so
    update the other treatments
266 //                          if (e2.eventType == eventType.TREATMENT)
267 //                              //TODO: this needs to subtract only the
     dead patients treatment time, not the events time
268 //                          e2.setTime(e2.getTime() + (e.getTime())-
    bigTime);
269 //                      }
270 //                  }
271 
272                 eventQ.managedRemove(i);
273                 return e.eventType; //returns what event was removed
    or null in fail case
274             }
275         }
```

```
276         return null;
277     }
278 
279     //generates new patient arrival from rate 3/hr
280     public static double TimeToArrive() throws Exception {
281         double deltaTime;
282         double bigX;
283         bigX = Math.random();
284         if (bigX > 0.9) {
285             bigX = Math.random();
286         }
287         deltaTime = -Math.log(1.0 - bigX) / .05;
288         return deltaTime;
289 
290     }//end timetoarrive
291 
292     //generates new treatment duration for a patients treatment event
293     public static double TimeToTreat(int a, double numDocs) {
294         double timeTreat;
295         double bigx = Math.random();
296         double rate = 0.0; //number of patients/hr
297         switch (a) {
298 
299             case 1://Heart
300                 rate = 2.0;
301                 break;
302             case 2://Gastro
303                 rate = 4.0;
304                 break;
305             case 3://Bleed
306                 rate = 6.0;
307                 break;
308             default:
309                 System.err.println("Wtf? This patient doesnt have an
    illness! Literally impossible.");
310                 System.exit(1);//there is a serious problem, exit
311         }
312 
313         timeTreat = 60 * Math.log(1 - bigx) / (-rate * numDocs);
314         return timeTreat;
315     }//end timetoTreat
316 
317 
318 }//end EventBasedClinicSimulation
319 
320 
    /*==================================================================
    ================================
```

```
321 Patient class
322 represents a patient at the clinic
323 Has Patient ID, Type of ailment, and arrival, wait, and total time
324
    ================================================================
    ================================*/
325 class Patient implements Comparable {
326     protected int ailmentType;  //1= heart,2=gastro, 3=bleeding
327
328     //arrival time, time waited, time till death, time in system
329     protected double tArrive;
330     protected double tTreat; //how long ilness will take to treat
331     protected double tWait;
332     protected double tDeath;
333     protected int myDeath;
334     protected int ID; //patient ID
335
336     public Patient(int ID, double bigTime) {
337         this.ID = ID;
338         this.myDeath = ID;
339         tWait = tTreat = 0;//defualt these to zero for safety
340         this.ailmentType = setAilmentType();
341         tArrive = bigTime;
342         settDeath(); //generate patient death time
343
344     }
345
346     public double gettTreat() {
347         return tTreat;
348     }
349
350     public void settTreat(double tTreat) {
351         this.tTreat = tTreat;
352     }
353
354     public int getID() {
355         return ID;
356     }
357
358     public int getMyDeath() {
359         return myDeath;
360     }
361
362     public int getAilmentType() {
363         return ailmentType;
364     }
365
366     private int setAilmentType() {
```

```
367         int r;
368 
369         int x = ((int) (Math.random() * 10));
370         if (x <= 3) {
371             r = 1; //Heart
372         } else if (x <= 5) {
373             r = 2;//Gastro
374         } else r = 3;//Bleed
375         return r;
376     }
377 
378 
379     public double gettArrive() {
380         return tArrive;
381     }
382 
383     public void settArrive(double tArrive) {
384         this.tArrive = tArrive;
385     }
386 
387     public double gettWait() {
388         return tWait;
389     }
390 
391     public void settWait(double tWait) {
392         this.tWait = tWait;
393     }
394 
395     public double gettDeath() {
396         return tDeath;
397     }
398 
399     private void settDeath() {
400         double dTimer;
401         int t = getAilmentType();
402         double mu = 0.0, sigma = 0.0;
403         switch (t) {
404 
405             case 1:
406                 mu = 10;
407                 sigma = 35;
408                 break;
409             case 2:
410                 mu = 30;
411                 sigma = 80;
412                 break;
413             case 3:
414                 mu = 20;
```

```
415                 sigma = 65;
416                 break;
417             default:
418                 System.err.println("Wtf? This patient doesnt have an illness and expects to die!");
419                 System.exit(1);//there is a serious problem, exit
420         }
421 
422         dTimer = sigma * (Math.random()) + mu;
423         tDeath = tArrive + dTimer;
424     }
425 
426     @Override
427 
428     public int compareTo(Object o) {
429         if (gettArrive() > ((Patient) o).gettArrive()) {
430             return 1;
431         } else if (gettArrive() < ((Patient) o).gettArrive()) {
432             return -1;
433         } else {
434             return 0;
435         }
436 
437     }
438 }
439 
440 
    /*================================================================================
441 GenericManager class
442 represents an array list of objects that can be compared to each other
443 objects can be added first, last, or added in order by comparator
444 mostly a copy of Kent Pickett's code - If it ain't broke don't fix it.
445 
    ================================================================================*/
446 class GenericManager<T extends Comparable<? super T>> {
447 
448     protected ArrayList<T> list = new ArrayList<>();
449     protected int count; //items in the arraylist
450 
451     //generic constructor
452     public GenericManager() {
453         //initialize to 0
454         this.count = 0;
455     }
```

```
456
457     public int getCount() {
458         return count;
459     }
460
461     public int addFront(T x) {
462         list.add(0, x);
463         count++;
464         return count;
465     }
466
467     public int addEnd(T x) {
468         list.add(count++, x);
469         return count;
470     }
471
472     //adds object x to the list a the position determined by its
    comparator
473     public int addInOrder(T x) {
474         int i;
475         if ((count == 0)
476                 || ((x.compareTo(list.get(0)) == -1)
477                 || x.compareTo(list.get(0)) == 0)) {
478             //object goes at the front of the list
479             list.add(0, x);
480         } else if ((x.compareTo(list.get(count - 1)) == 1)
481                 || (x.compareTo(list.get(count - 1)) == 0)) {
482             //object goes at the end of the list
483             list.add(count, x);
484         } else {
485             //object goes somewhere in the middle of the list
486             i = 0;
487             //compare x with the list from start until x > the
    current item
488             while ((i < count) && (x.compareTo(list.get(i)) == 1)) i
    ++;
489             //add x in its place after the current item
490             list.add(i, x);
491         }
492         count++;
493         return count;
494     }
495
496     public T getValue(int i) {
497         if (i < count) {
498             return list.get(i);
499         } else {
500             System.err.println(String.format("Attempted to get value
```

```
500 from a position that doesn't exist: %d", i));
501             return list.get(0);//default case
502         }
503     }
504 
505     //basic generic sorting method, uses object's compareTo
506     public void sort() {
507         T xsave, ysave, a, b;
508         int isw = 1; //is the list sorting
509         int xlast = list.size();
510         while (isw == 1) {
511             isw = 0;
512             for (int i = 0; i <= xlast - 2; i++) {
513                 a = list.get(i);
514                 b = list.get(i + 1);
515                 switch (a.compareTo(b)) {
516                     case 1://already sorted, break
517                         break;
518                     case -1://objects out of order, sort
519                         xsave = list.get(i);
520                         ysave = list.get(i + 1);
521                         list.remove(i);
522                         list.add(i, ysave);
523                         list.remove(i + 1);
524                         list.add(i + 1, xsave);
525                         isw = 1;
526                         break;
527                     default://objects assumed to be equal
528                 }
529             }
530         }
531     }//end of sorting method
532 
533     //removes item at specified index and decrements the count
534     public void managedRemove(int i) {
535         if ((i >= 0) && (i <= count - 1)) {
536             list.remove(i);
537             count--;
538         }
539     }
540 
541 }//end of generic mgr
542 
543 
    /*=================================================================
    ================================
544 Event class
545 represents event type, when the event occurs, and which patient it
```

```
belongs to
modified from Kent Pickett's code

================================================================================*/
class Event implements Comparable {

    protected EventType eventType; // event type
    protected int patient; // which patient this event belongs to
    private double time; //when this even occurs

    public Event(double time, EventType eventType, int patient) {
        this.eventType = eventType;
        this.time = time;
        this.patient = patient;
    }

    @Override
    //compares based on the events' times
    public int compareTo(Object o) {
        if (getTime() > ((Event) o).getTime()) {
            return 1;
        } else if (getTime() < ((Event) o).getTime()) {
            return -1;
        } else {
            return 0;
        }

    }

    public int getPatient() {
        return patient;
    }

    public EventType getEventType() {
        return eventType;
    }

    public double getTime() {
        return time;
    }

    public void setTime(double time) {
        this.time = time;
    }
}//end of Event

```