```
1  package pkgCOVID19;
2
3  /*****
4  * Class:  COVID19
5  * Version/Creation Date:  v1.0.1/20200328
6  * Programmer:  Garrett A. Hughes/ModelingComplexSystems.net
7  *
8  * Purpose:  Create a discrete event Systems Dynamics model of a COVID-19
9  *           epidemic in a general population
10 *
11 * FYI: See Featured Article on website ModelingComplexSystems.net
12 *       COVID-19 Dynamics
13 *       Part One: Infection Dynamics
14 *       Author: Garrett A. Hughes
15 *
16 *
17 *****/
18
19 import java.time.*;
20 import java.util.*;
21 import java.lang.Long;
22 import static java.lang.Math.random;
23 import static java.lang.Math.floor;
24 import static java.lang.Math.round;
25 import java.io.*;
26 import java.nio.charset.StandardCharsets;
27 import java.math.*;
28
29 class COVID19
30 {
31   // INSTANCE FIELD(S)
32     static String program = "COVID19";
33     static String version = "v1.0.1";
34
35     static Scanner sc1 = new Scanner(System.in);
36
37     static int minIncPeriod = 0;   // minimum incubation period [days]
38     static int maxIncPeriod = 0;   // maximum incubation period [days]
39     static int intRnd;             // integer random number mapped to range
40     static int time         = 0;   // time [days]
41     static int timeHorizon = 20;   // length of run [days]
42
43     static long popTot = 0;        // total population [persons]
44     static long popInc = 0;        // incubating population [persons]
45     static long popSus = 0;        // susceptible population [persons]
46     static long popPrs = 0;        // presenting population [persons]
47     static long popShl = 0;        // sheltered population [persons]
48     static long popRcv = 0;        // recovered population [persons]
49     static long infR   = 0;        // infection rate [persons/day]
50     static long prsR   = 0;        // presentation rate [persons/day]
51
52     static double MAX_infMul = 0.0;  // maximum infection multiplier
   [(persons/day)/person]
53     static double infMul     = 0.0;  // infection multiplier [(persons/day)/person]
54     static double Pinf       = 0.0;  // probability of infection [dimensionless]
55
56
57     // CONSTRUCTOR(S)
58
59
60
61   // METHOD(S)
62
63     static private int get_timeHorizon()
64     {
65      boolean invalidInput = false;
66         String nextToken = "";
67         do
68         {
```

```
69              invalidInput = false;
70              try
71              {
72                  System.out.println("\nEnter:");
73                  System.out.print ("   time horizon (timeHorizon)? ");
74                  timeHorizon = sc1.nextInt();  //read an integer
75              }
76              catch (InputMismatchException e)
77              {
78                  // clear bad token from System.in
79                  nextToken = sc1.nextLine();
80                  System.out.println("\n*****Invalid: value must be an integer");
81                  invalidInput = true;
82              }
83              // additional tests
84              if(invalidInput==false &&   timeHorizon < 0)
85                  {
86                      invalidInput = true;
87                      System.out.println("\n*****Invalid: value must be >= 0");
88                  };
89
90          } while ( invalidInput );
91          System.out.printf("\nYou entered:\n   time horizon = %,d\n", timeHorizon);
92          return timeHorizon;
93      }
94
95
96
97      static private int get_minIncPeriod()
98      {
99       boolean invalidInput = false;
100         String nextToken = "";
101         do
102         {
103             invalidInput = false;
104             try
105             {
106                 System.out.println("\nEnter:");
107                 System.out.print ("   minimum incubation period (minIncPeriod)? ");
108                 minIncPeriod = sc1.nextInt();  //read an integer
109             }
110             catch (InputMismatchException e)
111             {
112                 // clear bad token from System.in
113                 nextToken = sc1.nextLine();
114                 System.out.println("\n*****Invalid: value must be an integer");
115                 invalidInput = true;
116             }
117             // additional tests
118             if(invalidInput==false &&   minIncPeriod < 1)
119                 {
120                     invalidInput = true;
121                     System.out.println("\n*****Invalid: value must be >= 1");
122                 };
123
124         } while ( invalidInput );
125         System.out.printf("\nYou entered:\n   minimum incubation period = %,d\n",
    minIncPeriod);
126         return minIncPeriod;
127     }
128
129
130
131     static private int get_maxIncPeriod()
132     {
133      boolean invalidInput = false;
134         String nextToken = "";
135         do
136         {
```

```
137             invalidInput = false;
138             try
139             {
140                 System.out.println("\nEnter:");
141                 System.out.print ("   maximum incubation period (maxIncPeriod)? ");
142                 maxIncPeriod = sc1.nextInt();  //read an integer
143             }
144             catch (InputMismatchException e)
145             {
146                 // clear bad token from System.in
147                 nextToken = sc1.nextLine();
148                 System.out.println("\n*****Invalid: value must be an integer");
149                 invalidInput = true;
150             }
151             // additional tests
152             if(invalidInput==false &&   maxIncPeriod < minIncPeriod)
153                 {
154                     invalidInput = true;
155                     System.out.println("\n*****Invalid: value must be >= minimum
    incubation period");
156                 };
157 
158         } while ( invalidInput );
159         System.out.printf("\nYou entered:\n   maximum incubation period = %,d\n",
    maxIncPeriod);
160         return maxIncPeriod;
161     }
162 
163 
164 
165     static private long get_popTot()
166     {
167         boolean invalidInput = false;
168         String nextToken = "";
169         do
170         {
171             invalidInput = false;
172             try
173             {
174                 // Get total population: popTot
175                 System.out.println("\nEnter:");
176                 System.out.print ("   total population (popTot)? ");
177                 popTot = sc1.nextLong();  //read in a 64 bit integer
178             }
179             catch (InputMismatchException e)
180             {
181                 // clear bad token from System.in
182                 nextToken = sc1.nextLine();
183                 System.out.println("\n*****Invalid: value must be an integer");
184                 invalidInput = true;
185             }
186             // additional test
187             if(invalidInput==false && popTot <0)
188                 {
189                     invalidInput = true;
190                     System.out.println("\n*****Invalid: value must be >= 0");
191                 };
192         } while ( invalidInput );
193         System.out.printf("\nYou entered:\n   popTot = %,d\n", popTot);
194         return popTot;
195     }
196 
197 
198 
199     static private double get_MAX_infMul()
200     {
201 
202      /*****************
203       * this is the maximum number of people that a single individual from the
```

```
204         * incubating population - popInc - can infect in one day.
205         * units are [ (persons/day)/person ]
206         *
207         ******************/
208
209         boolean invalidInput = false;
210            String nextToken = "";
211            do
212            {
213                invalidInput = false;
214                try
215                {
216                    // Get maximum infection multiplier: max_infMul;
217                    System.out.println("\nEnter:");
218                    System.out.print ("   maxmum infection multiplier (MAX_infMul)? ");
219                    MAX_infMul = sc1.nextDouble();  //read in a double
220                }
221                catch (InputMismatchException e)
222                {
223                    // clear bad token from System.in
224                    nextToken = sc1.nextLine();
225                    System.out.println("\n*****Invalid: value must be a double");
226                    invalidInput = true;
227                }
228                // additional test
229                if(invalidInput==false &&  MAX_infMul < 0)
230                    {
231                        invalidInput = true;
232                        System.out.println("\n*****Invalid: value must be >= 0");
233                    };
234            } while ( invalidInput );
235            System.out.printf("\nYou entered:\n   MAX_infMul = %,5.2f\n", MAX_infMul);
236            return MAX_infMul;
237        }
238
239
240
241         static private long get_popInc()
242        {
243         boolean invalidInput = false;
244            String nextToken = "";
245            do
246            {
247                invalidInput = false;
248                try
249                {
250                    // Get initial value of incubating population: popInc;
251                    System.out.println("\nEnter:");
252                    System.out.print ("   incubating population (popInc)? ");
253                    popInc = sc1.nextLong();  //read in a 64 bit integer
254                }
255                catch (InputMismatchException e)
256                {
257                    // clear bad token from System.in
258                    nextToken = sc1.nextLine();
259                    System.out.println("\n*****Invalid: value must be an integer");
260                    invalidInput = true;
261                }
262                // additional tests
263                if(invalidInput==false &&   popInc < 0)
264                    {
265                        invalidInput = true;
266                        System.out.println("\n*****Invalid: value must be >= 0");
267                    };
268                if(invalidInput==false && popInc > popTot)
269                    {
270                        invalidInput = true;
271                        System.out.println("\n*****Invalid: value must be <= popTot");
272                    }
```

```
273          } while ( invalidInput );
274          System.out.printf("\nYou entered:\n   popInc = %,d\n", popInc);
275          return popInc;
276      }
277
278
279
280      static public int get_Random(int minIncPeriod, int maxIncPeriod)
281      {
282          double min = (double)minIncPeriod;
283          double max = (double)maxIncPeriod;
284
285          double rnd = random();
286          double r2 = (rnd * (max +1 -min) + min);  // convert to range [min, max]
287          double rf = floor(r2);                    // convert to a whole number <= r2
288          int r3 = (int)rf;                         // covert to an integer
289          return (r3);
290      }
291
292
293
294
295
296
297
298
299      /*_____________________________MAIN_____________________________*/
300
301    public static void main(String[] args)
302    {
303          LocalDateTime LDT = LocalDateTime.now();
304          System.out.println("\nPROGRAM: " + program + "  " +version + "  " + LDT +
     "\n" );
305
306          long sum = 0;
307          File file = new File("COVID19.txt");
308
309          /************BEGIN: INITIALIZATION OF THE MODEL***********/
310
311               timeHorizon     = get_timeHorizon();
312               minIncPeriod    = get_minIncPeriod();
313               maxIncPeriod    = get_maxIncPeriod();
314               popTot          = get_popTot();
315               popInc          = get_popInc();
316               popSus = popTot - popInc;
317
318               long [] incQue = new long[maxIncPeriod+1];   // incubating pop queue
319
320               MAX_infMul  = get_MAX_infMul();
321
322               // Initialize the incubating population queue to zero
323               for(int i=0; i<= maxIncPeriod; i++)
324                 incQue[i] = 0;
325
326               // Distribute initial arrivals to the popInc queue
327               for(long i=0; i<popInc; i++) // for each arrival
328               {
329                   intRnd = get_Random(minIncPeriod, maxIncPeriod);
330                   incQue[intRnd]+=1;
331               };
332
333                   /*****
334                   * With service times of minIncPeriod to maxIncPeriod
335                   * Values of intRnd are currently drawn from a uniform distribution
336
337                   * We need to set the values of popInc and Max_InfMul
338                   * large enough to yield an infR >= 1 to
339                   * have the contagion kick off
340
```

```
341
342                 * Select a random integer in the interval [minIncPeriod,
    maxIncPeriod]
343                 * getting a random value the service time
344
345                 * Put that person in an array position whose value
346                 * is equal to their associated service
    time
347
348                 * Later we treat this service queue like a register and
349                 * shift the contents toward the zeroth position each day
350                 * effectively reducing the occupants' incubation period by one day
351
352                 * We are currently using the uniform distribution to generate
353                 * random integers for the incubation period. You can change this by
354                 * introducing a different function and renaming it get_ExpRandom or
355                 * something like that to reflect the distributon you want to use.
356                 * You may have to increase the size of the incQue to hold additional
357                 * days if the new distribution has a longer tail or shorter
358                 * minimum incubation period
359                 *****/
360
361         /**********END: INITIALIZATION  OF THE MODEL**********/
362
363
364
365         /**********BEGIN: SIMULATION**********/
366
367         // Start by opening a file to store time dependent state variables
368         try (FileWriter fw = new FileWriter(file);
369             BufferedWriter bf = new BufferedWriter(fw);
370             PrintWriter out = new PrintWriter(bf) )
371         {
372             out.printf(" time          popTot           popSus
    popInc           popPrs       Pinf             infR  infMul\n");
373
374             do  // BEGIN: OUTER LOOP OF SIMULATION
375             {
376                 // Indicate state variable at beginning of day (24 hour period)
377                 out.printf("  %3d  %,14d   %,14d   %,14d   %,14d    %6.5f   %,14d
    %6.4f\n", time, popTot, popSus, popInc, popPrs, Pinf, infR, infMul);
378
379                 // COMPUTE RATES AND AUXILIARIES AT THE BEGINNING OF THE DAY
380
381
382
383
384                 // Compute the infection multiplier (infMul) [ (persons/day)/person ]
385                 infMul = MAX_infMul * (double)popSus/(double)popTot;
386
387                 /*****
388                 * Note: The infection multiplier varies
389                 * as the ratio of the susceptible population to the total
    population decreases.
390                 * We assume that each person who is infected but asymptomatic,
391                 * affects a number of the susceptible poulation
392                 * equal to the infection multiplier
393                 *****/
394
395
396                 // Compute the infection rate (infR) [persons/day]
397                 infR = Math.round((infMul * (double)popInc));
398
399                 /*****
400                 * Note: We multiply the total number of members of the
401                 * incubating population by the infection multiplier to get
402                 * the number infected on any given day
403                 *****/
404
```

```
405
406                 // Compute the probability of becoming infected on this day
407                 Pinf = infR/(double)popSus;
408
409                 // Advance time to the end of the day
410                 for (int i = 0; i < (maxIncPeriod); i++)
411                     incQue[i] = incQue[i+1];
412
413
414
415                 // COMPUTE NEW LEVELS BASED ON THE RATES FOR THAT DAY
416
417                 // get the presentation rate
418                 prsR = incQue[0];           // Get increment to the presenting
    population
419                 incQue[0] = 0;              // Clear the head of the queue
420
421                 /*****
422                 * Note: The presentation rate is determined by the number of people
423                 * at the head of the incubation queue
424                 *****/
425
426                 // zero out the tail
427                 incQue[maxIncPeriod] = 0;
428
429
430                 // Compute new susceptible population
431                 popSus = popSus - Math.round(infR);
432
433                 /*****
434                 * Note: the Math.round() method rounds to the nearest long.
435                 * Don't cast infR to a long this way,  (long)infR
436                 * because it will truncate the double!!  Nor use (int) either.
437                 *****/
438
439
440                 // Update incubating population with infected individuals
441                 for(int i=0; i<infR; i++) // for each arrival
442                 {
443                     intRnd = get_Random(minIncPeriod, maxIncPeriod);
444                     incQue[intRnd]+=1;
445                 };
446
447                 /*****
448                 * Note: Adds new arrivals to the incubating population
449                 * with randomly chosen incubation periods
450                 * This could take a little time for a billion entries
451                 *****/
452
453
454                 // Compute new size of the incubating population
455                 sum = 0;
456
457                 for (int i = 1; i<= maxIncPeriod; i++)
458                 {
459                     sum = sum + incQue[i];
460
461                 };
462
463                 popInc = sum;
464
465
466                 // Compute the new presenting population
467                 popPrs = popPrs + prsR;
468
469                 // Compute checksum of total population
470                 // Should always be the same name as the initial total population
471                 popTot= popSus + popInc + popPrs + popShl + popRcv;
472
```

```
473
474
475                 // Increment time by one day
476                 time+=1;
477
478             } while (time <= timeHorizon);  // END: OUTER LOOP OF SIMULATION
479
480             // close BufferedWriter and PrintWriter
481             bf.close();
482             out.close();
483             LDT = LocalDateTime.now();
484             System.out.println("Successful completion at: " + LDT);
485         }  // END - try
486
487         catch (IOException e)
488         {
489
490         }  // END - CATCH
491
492     }  // END - Main
493
494 }  // END - class COVID19
495
496
497
498
```