```java
package pkgCOVID19;

/*****
* Class:  COVID19
* Version/Creation Date:  v1.0.1/20200328
* Programmer:  Garrett A. Hughes/ModelingComplexSystems.net
*
* Purpose:  Create a discrete event Systems Dynamics model of a COVID-19
*           epidemic in a general population
*
* FYI: See Featured Article on website ModelingComplexSystems.net
*       COVID-19 Dynamics
*       Part One: Infection Dynamics
*       Author: Garrett A. Hughes
*
*
*****/

import java.time.*;
import java.util.*;
import java.lang.Long;
import static java.lang.Math.random;
import static java.lang.Math.floor;
import static java.lang.Math.round;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.math.*;

class COVID19
{
  // INSTANCE FIELD(S)
    static String program = "COVID19";
    static String version = "v1.0.1";

    static Scanner sc1 = new Scanner(System.in);

    static int minIncPeriod = 0;   // minimum incubation period [days]
    static int maxIncPeriod = 0;   // maximum incubation period [days]
    static int intRnd;             // integer random number mapped to range
    static int time         = 0;   // time [days]
    static int timeHorizon = 20;   // length of run [days]

    static long popTot = 0;        // total population [persons]
    static long popInc = 0;        // incubating population [persons]
    static long popSus = 0;        // susceptible population [persons]
    static long popPrs = 0;        // presenting population [persons]
    static long popShl = 0;        // sheltered population [persons]
    static long popRcv = 0;        // recovered population [persons]
    static long infR   = 0;        // infection rate [persons/day]
    static long prsR   = 0;        // presentation rate [persons/day]

    static double MAX_infMul = 0.0;  // maximum infection multiplier [(persons/day)/person]
    static double infMul     = 0.0;  // infection multiplier [(persons/day)/person]
    static double Pinf       = 0.0;  // probability of infection [dimensionless]


    // CONSTRUCTOR(S)



  // METHOD(S)

    static private int get_timeHorizon()
    {
     boolean invalidInput = false;
        String nextToken = "";
        do
        {
            invalidInput = false;
            try
            {
                System.out.println("\nEnter:");
                System.out.print ("   time horizon (timeHorizon)? ");
                timeHorizon = sc1.nextInt();  //read an integer
            }
            catch (InputMismatchException e)
            {
                // clear bad token from System.in
                nextToken = sc1.nextLine();
                System.out.println("\n*****Invalid: value must be an integer");
                invalidInput = true;
            }
            // additional tests
            if(invalidInput==false &&   timeHorizon < 0)
                {
                    invalidInput = true;
                    System.out.println("\n*****Invalid: value must be >= 0");
                };

        } while ( invalidInput );
        System.out.printf("\nYou entered:\n   time horizon = %,d\n", timeHorizon);
        return timeHorizon;
    }



    static private int get_minIncPeriod()
    {
     boolean invalidInput = false;
        String nextToken = "";
        do
        {
            invalidInput = false;
            try
            {
                System.out.println("\nEnter:");
                System.out.print ("   minimum incubation period (minIncPeriod)? ");
                minIncPeriod = sc1.nextInt();  //read an integer
            }
            catch (InputMismatchException e)
            {
                // clear bad token from System.in
                nextToken = sc1.nextLine();
                System.out.println("\n*****Invalid: value must be an integer");
                invalidInput = true;
            }
            // additional tests
            if(invalidInput==false &&   minIncPeriod < 1)
                {
                    invalidInput = true;
                    System.out.println("\n*****Invalid: value must be >= 1");
                };

        } while ( invalidInput );
        System.out.printf("\nYou entered:\n   minimum incubation period = %,d\n", minIncPeriod);
        return minIncPeriod;
    }



    static private int get_maxIncPeriod()
    {
     boolean invalidInput = false;
        String nextToken = "";
        do
        {
            invalidInput = false;
            try
            {
                System.out.println("\nEnter:");
                System.out.print ("   maximum incubation period (maxIncPeriod)? ");
                maxIncPeriod = sc1.nextInt();  //read an integer
            }
            catch (InputMismatchException e)
            {
                // clear bad token from System.in
                nextToken = sc1.nextLine();
                System.out.println("\n*****Invalid: value must be an integer");
                invalidInput = true;
            }
            // additional tests
            if(invalidInput==false &&   maxIncPeriod < minIncPeriod)
                {
                    invalidInput = true;
                    System.out.println("\n*****Invalid: value must be >= minimum
incubation period");
                };

        } while ( invalidInput );
        System.out.printf("\nYou entered:\n   maximum incubation period = %,d\n",
maxIncPeriod);
        return maxIncPeriod;
    }



    static private long get_popTot()
    {
        boolean invalidInput = false;
        String nextToken = "";
        do
        {
            invalidInput = false;
            try
            {
                // Get total population: popTot
                System.out.println("\nEnter:");
                System.out.print ("   total population (popTot)? ");
                popTot = sc1.nextLong();  //read in a 64 bit integer
            }
            catch (InputMismatchException e)
            {
                // clear bad token from System.in
                nextToken = sc1.nextLine();
                System.out.println("\n*****Invalid: value must be an integer");
                invalidInput = true;
            }
            // additional test
            if(invalidInput==false && popTot <0)
                {
                    invalidInput = true;
                    System.out.println("\n*****Invalid: value must be >= 0");
                };
        } while ( invalidInput );
        System.out.printf("\nYou entered:\n   popTot = %,d\n", popTot);
        return popTot;
    }



    static private double get_MAX_infMul()
    {

     /****************
     * this is the maximum number of people that a single individual from the
     * incubating population - popInc - can infect in one day.
     * units are [ (persons/day)/person ]
     *
     *****************/

     boolean invalidInput = false;
        String nextToken = "";
        do
        {
            invalidInput = false;
            try
            {
                // Get maximum infection multiplier: max_infMul;
                System.out.println("\nEnter:");
                System.out.print ("   maxmum infection multiplier (MAX_infMul)? ");
                MAX_infMul = sc1.nextDouble();  //read in a double
            }
            catch (InputMismatchException e)
            {
                // clear bad token from System.in
                nextToken = sc1.nextLine();
                System.out.println("\n*****Invalid: value must be a double");
                invalidInput = true;
            }
            // additional test
            if(invalidInput==false &&  MAX_infMul < 0)
                {
                    invalidInput = true;
                    System.out.println("\n*****Invalid: value must be >= 0");
                };
        } while ( invalidInput );
        System.out.printf("\nYou entered:\n   MAX_infMul = %,5.2f\n", MAX_infMul);
        return MAX_infMul;
    }



     static private long get_popInc()
    {
     boolean invalidInput = false;
        String nextToken = "";
        do
        {
            invalidInput = false;
            try
            {
                // Get initial value of incubating population: popInc;
                System.out.println("\nEnter:");
                System.out.print ("   incubating population (popInc)? ");
                popInc = sc1.nextLong();  //read in a 64 bit integer
            }
            catch (InputMismatchException e)
            {
                // clear bad token from System.in
                nextToken = sc1.nextLine();
                System.out.println("\n*****Invalid: value must be an integer");
                invalidInput = true;
            }
            // additional tests
            if(invalidInput==false &&   popInc < 0)
                {
                    invalidInput = true;
                    System.out.println("\n*****Invalid: value must be >= 0");
                };
            if(invalidInput==false && popInc > popTot)
                {
                    invalidInput = true;
                    System.out.println("\n*****Invalid: value must be <= popTot");
                }
        } while ( invalidInput );
        System.out.printf("\nYou entered:\n   popInc = %,d\n", popInc);
        return popInc;
    }



    static public int get_Random(int minIncPeriod, int maxIncPeriod)
    {
        double min = (double)minIncPeriod;
        double max = (double)maxIncPeriod;

        double rnd = random();
        double r2 = (rnd * (max +1 -min) + min);  // convert to range [min, max]
        double rf = floor(r2);                    // convert to a whole number <= r2
        int r3 = (int)rf;                         // covert to an integer
        return (r3);
    }








    /*________________________MAIN_________________________*/

  public static void main(String[] args)
  {
       LocalDateTime LDT = LocalDateTime.now();
       System.out.println("\nPROGRAM: " + program + "  " +version + "  " + LDT +
"\n" );

       long sum = 0;
       File file = new File("COVID19.txt");

       /************BEGIN: INITIALIZATION OF THE MODEL***********/

            timeHorizon     = get_timeHorizon();
            minIncPeriod    = get_minIncPeriod();
            maxIncPeriod    = get_maxIncPeriod();
            popTot          = get_popTot();
            popInc          = get_popInc();
            popSus = popTot - popInc;

            long [] incQue = new long[maxIncPeriod+1];   // incubating pop queue

            MAX_infMul  = get_MAX_infMul();

            // Initialize the incubating population queue to zero
            for(int i=0; i<= maxIncPeriod; i++)
              incQue[i] = 0;

            // Distribute initial arrivals to the popInc queue
            for(long i=0; i<popInc; i++) // for each arrival
            {
                intRnd = get_Random(minIncPeriod, maxIncPeriod);
                incQue[intRnd]+=1;
            };

                /*****
                * With service times of minIncPeriod to maxIncPeriod
                * Values of intRnd are currently drawn from a uniform distribution

                * We need to set the values of popInc and Max_InfMul
                * large enough to yield an infR >= 1 to
                * have the contagion kick off


                * Select a random integer in the interval [minIncPeriod, maxIncPeriod]
                * getting a random value the service time

                * Put that person in an array position whose value
                * is equal to their associated service time

                * Later we treat this service queue like a register and
                * shift the contents toward the zeroth position each day
                * effectively reducing the occupants' incubation period by one day

                * We are currently using the uniform distribution to generate
                * random integers for the incubation period. You can change this by
                * introducing a different function and renaming it get_ExpRandom or
                * something like that to reflect the distributon you want to use.
                * You may have to increase the size of the incQue to hold additional
                * days if the new distribution has a longer tail or shorter
                * minimum incubation period
                *****/

        /**********END: INITIALIZATION  OF THE MODEL**********/



        /**********BEGIN: SIMULATION**********/

        // Start by opening a file to store time dependent state variables
        try (FileWriter fw = new FileWriter(file);
            BufferedWriter bf = new BufferedWriter(fw);
            PrintWriter out = new PrintWriter(bf) )
        {
            out.printf(" time          popTot           popSus           popInc           popPrs       Pinf             infR  infMul\n");

            do  // BEGIN: OUTER LOOP OF SIMULATION
            {
                // Indicate state variable at beginning of day (24 hour period)
                out.printf("  %3d  %,14d   %,14d   %,14d   %,14d    %6.5f   %,14d   %6.4f\n", time, popTot, popSus, popInc, popPrs, Pinf, infR, infMul);

                // COMPUTE RATES AND AUXILIARIES AT THE BEGINNING OF THE DAY




                // Compute the infection multiplier (infMul) [ (persons/day)/person ]
                infMul = MAX_infMul * (double)popSus/(double)popTot;

                /*****
                * Note: The infection multiplier varies
                * as the ratio of the susceptible population to the total population decreases.
                * We assume that each person who is infected but asymptomatic,
                * affects a number of the susceptible poulation
                * equal to the infection multiplier
                *****/


                // Compute the infection rate (infR) [persons/day]
                infR = Math.round((infMul * (double)popInc));

                /*****
                * Note: We multiply the total number of members of the
                * incubating population by the infection multiplier to get
                * the number infected on any given day
                *****/


                // Compute the probability of becoming infected on this day
                Pinf = infR/(double)popSus;

                // Advance time to the end of the day
                for (int i = 0; i < (maxIncPeriod); i++)
                    incQue[i] = incQue[i+1];



                // COMPUTE NEW LEVELS BASED ON THE RATES FOR THAT DAY

                // get the presentation rate
                prsR = incQue[0];           // Get increment to the presenting population
                incQue[0] = 0;              // Clear the head of the queue

                /*****
                * Note: The presentation rate is determined by the number of people
                * at the head of the incubation queue
                *****/

                // zero out the tail
                incQue[maxIncPeriod] = 0;


                // Compute new susceptible population
                popSus = popSus - Math.round(infR);

                /*****
                * Note: the Math.round() method rounds to the nearest long.
                * Don't cast infR to a long this way,  (long)infR
                * because it will truncate the double!!  Nor use (int) either.
                *****/


                // Update incubating population with infected individuals
                for(int i=0; i<infR; i++) // for each arrival
                {
                    intRnd = get_Random(minIncPeriod, maxIncPeriod);
                    incQue[intRnd]+=1;
                };

                /*****
                * Note: Adds new arrivals to the incubating population
                * with randomly chosen incubation periods
                * This could take a little time for a billion entries
                *****/


                // Compute new size of the incubating population
                sum = 0;

                for (int i = 1; i<= maxIncPeriod; i++)
                {
                    sum = sum + incQue[i];

                };

                popInc = sum;


                // Compute the new presenting population
                popPrs = popPrs + prsR;

                // Compute checksum of total population
                // Should always be the same name as the initial total population
                popTot= popSus + popInc + popPrs + popShl + popRcv;

```

```


               // Increment time by one day
               time+=1;

           } while (time <= timeHorizon);  // END: OUTER LOOP OF SIMULATION

           // close BufferedWriter and PrintWriter
           bf.close();
           out.close();
           LDT = LocalDateTime.now();
           System.out.println("Successful completion at: " + LDT);
       }  // END - try

       catch (IOException e)
       {

       }  // END - CATCH

   }  // END - Main

}  // END - class COVID19
```