```java
// $Header: /home/cvs/t21617/samuel.riedo/s3/RollerCoaster.java,v 1.7 2017-05-29 05:49:48 samuel.riedo Exp $

import java.util.*;
import java.net.Socket;
import java.net.ServerSocket;
import java.net.InetAddress;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

/*
* The goal of this class is to make a simple simulation of an amusement park.
* There is a Wagon with a capacity of "csize" guys, and "pnum" visitors in the park.
* Each visitor visit the park, wait for a ride and start again "piteration"'s time.
* The wagon wait to be full, go around the track, unload passengers and start again until
* there are less visitors in the park than it's capacity. When this happens, the wagon
* waits for the remaining passengers to board, goes around the track, and waits for
* the passengers to leave the wagon. The passengers can no longer board as the park
* is closing. Once the wagon and passenger processes have ended, the simulation ends.
*
* Program arguments:
*  1: pnum
*  2: piterations
*  3: csize
*
*/
public class RollerCoaster{

    static int pnum = 17;                       // Visitor(s) at the park.
    static int piterations = 41;                // Iteration(s) per visitor.
    static int csize = 5;                       // Wagon capacity (people it can have on board).
    static int visitorRunningThread = 0;        // Number of visitor running thread(s).
    static Thread threadTable[];                // Countain all visitor(s) and wagon threads.
    static boolean simulationEnd = false;       // When true, stop the simulation.
    static final int PORT_NUMBER = 4356;        // Socket port number.

    /*
    * Program entry point. Simulate the behavioral of visitors in a park with a roller coaster. Visitor visit
    * the park and wait to take rides in the wagon.
    * @param
    *  1: pnum
    *  2: piterations
    *  3: csize
    */
    public static void main(String... args){
        int argsl = args.length;                        // Main programm arguments.
        switch (argsl){
            case 3:
                csize = Integer.valueOf(args[--argsl]);
            case 2:
                piterations = Integer.valueOf(args[--argsl]);
            case 1:
                pnum = Integer.valueOf(args[--argsl]);
        }
        threadTable = new Thread[pnum + 1];              // + 1 for the wagon.
        threadsStartup();
        waitOnThread();
        System.out.printf("End Simulation.\n");
    }

    /*
    * Create a Wagon thread and pnum Visitor thread(s).
    * Store them in threadTable, then shuffle it and start
    * all threads.
    */
    private static void threadsStartup(){
        threadTable[0] = new Thread(new Wagon(0, csize));

        for (int i = 1; i < threadTable.length; i++){
            threadTable[i] = new Thread(new Visitor(i, piterations));
            visitorRunningThread++;
        }

        Collections.shuffle(Arrays.asList(threadTable));
        for (int i = 0; i < threadTable.length; i++){
            threadTable[i].start();
        }
    }

    /*
    * Wait on all visitor's threads and the wagon thread to be terminated, then,
    * return.
    */
    private static void waitOnThread(){
        try{
            for (int i = 0; i < threadTable.length; i++){
                threadTable[i].join();
            }
        }
        catch(InterruptedException e){
            System.out.printf("Exception while waiting on thread to terminat.\n");
            e.printStackTrace();
        }
    }

    /*
    * Simulate the behavioral of a visitor doing the following tasks:
    *   - Visit the park.
    *   - Wait to embark the wagon.
    *   - Take a ride and leave the wagon.
    * This process is done piter's time.
    */
    static class Visitor implements Runnable{

        private int id = 0;                     // Unique thread ID.
        private int iterations = 0;             // Visitor iteration already done.
        private Socket socket;                  // Visitor socket.
        private DataInputStream input;          // Input stream. Get data from Wagon.
        private DataOutputStream output;        // Output stream. Send data to Wagon.
        static final int MAX_SLEEP_TIME = 30 ;  // Max visitor thread sleep time (in ns).

        /*
        * Contructor, set visitor id and iterations.
        * @param
        *   - int thread unique id.
        *   - int number of iterations.
        */
        public Visitor(int threadID, int it){
            this.id = threadID;
            this.iterations = it;
        }

        /*
        * Do the following tasks "this.iterations"' time(s):
        *   - Visit the park.
        *   - Wait to embark the wagon.
        *   - Take a ride and leave the wagon.
        */
        @Override
        public void run(){
            while(iterations>0 && !simulationEnd){
                this.iterations--;
                visitThePark();
                waitForWagon();
                leaveWagon();
            }
            System.out.printf("Visitor %d left the park, %d visitor(s) remaining in the park.\n",
                this.id, RollerCoaster.visitorRunningThread);
        }

        /*
        * Simulate a visit in the park with a thread.sleep.
        */
        private void visitThePark(){
            try{
                Thread.sleep(0, (int)(Math.random()*MAX_SLEEP_TIME));
            }
            catch (InterruptedException e){
                System.out.printf("Exception occurred while visitor %d was sleeping\n", this.id);
                e.printStackTrace();
            }
            System.out.printf("Visitor %d visited the park.\n", id);
        }

        /*
         * Wait to embark the wagon. This method communicate with the wagon using socket.
         */
        private void waitForWagon(){
            try{
                socket = new Socket(InetAddress.getLocalHost(), RollerCoaster.PORT_NUMBER); // Initialize communication.
                input = new DataInputStream(socket.getInputStream());
                output = new DataOutputStream(socket.getOutputStream());

                System.out.printf("Visitor %d wait for a ride.\n", this.id);
                output.writeInt(id);
                output.writeBoolean(true);                                                  // Visitor ready.
                output.writeInt(this.iterations);
                input.readBoolean();                                                        // Ride terminated.
            }
            catch (IOException e){
                System.out.printf("Exception occurred while visitor %d was on the ride.\n", this.id);
                e.printStackTrace();
            }
        }

        /*
         * When the ride is finished, the seat occuped by this visitor will be freed.
         */
        private void leaveWagon(){
            try{
                output.writeBoolean(true);
                System.out.printf("Visitor %d left wagon\n", this.id);

                input.close();              // Close communication.
                socket.close();
            }
            catch (IOException e){
                System.out.printf("Exception occurred while visitor %d tried to leave wagon.\n", this.id);
                e.printStackTrace();
            }
        }
    }

    /*
     * Simulate the behavioral of a wagon doing the following tasks:
     *      - Wait to be full of passengers.
     *      - Do a ride.
     *      - Wait to be empty.
     * These tasks are done while there is more visitors in the park than the wagon capacity.
     * When this occurs, the wagon do a last ride with the remaining visitor.
     */
    static class Wagon implements Runnable{

        private int id = 0;                 // Unique thread ID.
        private int capacity = 0;           // Wagon capacity (people it can have on board).
        private ServerSocket serverSocket;  // Server socket, accept connections.
        private Socket[] socket;            // Server/client socket table.
        private DataInputStream[] input;    // Input stream, one for each visitor.
        private DataOutputStream[] output;  // Output stream, one for each visitor.

        /*
         * Constructor, set wagon id and capacity.
         *
         * @param
         *   - int thread unique id.
         *   - int wagon's capacity.
         */
        public Wagon(int threadID, int csize){
            this.id = threadID;
            this.capacity = csize;
        }

        /*
         * Do the following tasks:
         *      - Wait to be full of passengers.
         *      - Do a ride.
         *      - Wait to be empty.
         * While there is more visitors in the park than the wagon capacity.
         * When this occurs, the wagon do a last ride with the remaining visitor.
         */
        @Override
        public void run(){
            initializeConnection();
            while(RollerCoaster.visitorRunningThread>= this.capacity){
                loadingPassengers();
                ride();
                unloadingPassangers();
            }
            simulationEnd = true;
            this.capacity = RollerCoaster.visitorRunningThread;     // Last ride.
            if(this.capacity!=0){
                loadingPassengers();
                ride();
                unloadingPassangers();
            }
            terminateConnection();
        }

        /*
         * Initialize server socket.
         */
        private void initializeConnection(){
            try {
                socket = new Socket[this.capacity];
                serverSocket = new ServerSocket(RollerCoaster.PORT_NUMBER);
                input = new DataInputStream[this.capacity];
                output = new DataOutputStream[this.capacity];
            }
            catch (IOException e){
                System.out.printf("Can't Initialize connectins for wagon %d.\n", this.id);
                e.printStackTrace();
            }
        }

        /*
         * Load passengers. This method leave when the amount of visitors on board is egal to
         * the capacity of the wagon or when all visitor in the park are on board.
         */
        private void loadingPassengers(){
            try{
                for(int i = 0; i<this.capacity; i++){
                    socket[i] = serverSocket.accept();                          // Initialize connection.
                    input[i] = new DataInputStream(socket[i].getInputStream());
                    output[i] = new DataOutputStream(socket[i].getOutputStream());

                    System.out.printf("Visitor %d boarded wagon %d.\n", input[i].readInt(), this.id);

                    input[i].readBoolean();                                     // Visitor ready.
                    if (input[i].readInt()==0)                                  // Visitor last iterations.
                        RollerCoaster.visitorRunningThread--;
                }
            }
            catch (IOException e){
                System.out.printf("Exception occurred while loading passengers on wagon %d.\n", this.id);
                e.printStackTrace();
            }
        }

        /*
         * The wagon go aroung the track. Visitor can't leave it before the end of the ride.
         */
        private void ride(){
            try{
                System.out.printf("Wagon %d start going aroung the track.\n", this.id);

                for(int i = 0; i<this.capacity; i++){
                    output[i].writeBoolean(true);
                }
            }
            catch (IOException e){
                System.out.printf("Exception occurred while wagon %d was going around the track.\n", this.id);
                e.printStackTrace();
            }
        }

        /*
         * Unload all wagon's passangers.
         */
        private void unloadingPassangers(){
            try{
                for(int i = 0; i<this.capacity; i++){
                    input[i].readBoolean();             // Wait for everybody to leave.
                }
                for(int i = 0; i<this.capacity; i++){
                    input[i].close();                   // Close communication.
                    output[i].close();
                    socket[i].close();
                }
                System.out.printf("Wagon %d succesfully unloaded all passengers.\n", this.id);
            }
            catch (IOException e){
                System.out.printf("Exception occurred while wagon %d was unloading passengers.\n", this.id);
                e.printStackTrace();
            }
        }

        /*
         * Terminate connection at the end of the simulation.
         */
        private void terminateConnection(){
            try {
                serverSocket.close();
            }
            catch (IOException e){
                System.out.printf("Exception occurred while wagon %d tried to close connection.\n", this.id);
                e.printStackTrace();
            }
        }
    }
}


/*
** $Log: RollerCoaster.java,v $
** Revision 1.7  2017-05-29 05:49:48  samuel.riedo
** Final version
** Some comments were modified to be more accurate. No code change except some typo modifications.
**
** Revision 1.6  2017-05-28 19:16:48  samuel.riedo
** Typo and 10 commandmends check.
**
** Revision 1.5  2017-05-25 14:58:43  samuel.riedo
** Add a join on all thread at the end of RollerCoaster main method. Thereby, "End Simulation" message
** is now properly displayed at the effective simulation's end.
**
** Revision 1.4  2017-05-25 14:49:08  samuel.riedo
** Bug corrected.
** When the wagon has done the last ride because the visitors in the park is lower than
** wagon capacity and the remaining visitor still have iterations to do, the visitors were blocked.
** To correct that, I created a new boolean "simulationEnd" that is set to true after the wagon's last
** ride. This condition is tested by visitor to do new iterations and so they are no longer
** blocked when there is no remaining wagon.
**
** Revision 1.3  2017-05-25 12:58:19  samuel.riedo
**
** First functionnal version.
** There was a bug with the end of the programm. At the beggining, I did it with a variable
** incremented when visitor's threads were created (before they were started) and i decremented
** the variable in visitor run method just before leaving the method.
** The problem was this variable is a critical section and was not protected. I could have used
** a mutex to protect it, but, as the TP is about socket, I used the following solution:
** When boarding, a visitor thread send remaining iterations number to the wagon. If this number
** is 0, then wagon thread decrement the variable. As there is only one wagon thread, there is no
** issue.
**
** Revision 1.2  2017-05-25 08:13:02  samuel.riedo
**
** Add Program skeleton. Create method, but they are unimplemented.
**
** Revision 1.1  2017-05-01 08:44:19  samuel.riedo
** Initial commit to test.
**
*/
```

```
java RollerCoaster
Visitor 3 visited the park.
Visitor 12 visited the park.
Visitor 17 visited the park.
Visitor 4 visited the park.
Visitor 16 visited the park.
Visitor 11 visited the park.
Visitor 5 visited the park.
Visitor 14 visited the park.
Visitor 13 visited the park.
Visitor 9 visited the park.
Visitor 6 visited the park.
Visitor 10 visited the park.
Visitor 8 visited the park.
Visitor 7 visited the park.
Visitor 15 visited the park.
Visitor 2 visited the park.
Visitor 1 visited the park.
Visitor 7 wait for a ride.
Visitor 11 wait for a ride.
Visitor 10 wait for a ride.
Visitor 2 wait for a ride.
Visitor 6 wait for a ride.
Visitor 8 wait for a ride.
Visitor 14 wait for a ride.
Visitor 5 wait for a ride.
Visitor 16 wait for a ride.
Visitor 17 wait for a ride.
Visitor 4 wait for a ride.
Visitor 3 wait for a ride.
Visitor 13 wait for a ride.
Visitor 9 wait for a ride.
Visitor 12 wait for a ride.
Visitor 7 boarded wagon 0.
Visitor 1 wait for a ride.
Visitor 15 wait for a ride.
Visitor 1 boarded wagon 0.
Visitor 11 boarded wagon 0.
Visitor 2 boarded wagon 0.
Visitor 8 boarded wagon 0.
Wagon 0 start going aroung the track.
Visitor 7 left wagon
Visitor 7 visited the park.
Visitor 7 wait for a ride.
Visitor 1 left wagon
Visitor 11 left wagon
Visitor 2 left wagon
Visitor 1 visited the park.
Visitor 11 visited the park.
Visitor 2 visited the park.
########################################..... 3018 lines skipped ......
########################################
Wagon 0 succesfully unloaded all passengers.
Visitor 2 boarded wagon 0.
Visitor 9 boarded wagon 0.
Visitor 17 boarded wagon 0.
Visitor 11 boarded wagon 0.
Visitor 14 boarded wagon 0.
Wagon 0 start going aroung the track.
Visitor 2 left wagon
Visitor 2 left the park, 6 visitor(s) remaining in the park.
Visitor 9 left wagon
Visitor 17 left wagon
Visitor 17 left the park, 6 visitor(s) remaining in the park.
Visitor 11 left wagon
Visitor 11 left the park, 6 visitor(s) remaining in the park.
Visitor 14 left wagon
Visitor 14 left the park, 6 visitor(s) remaining in the park.
Wagon 0 succesfully unloaded all passengers.
Visitor 4 boarded wagon 0.
Visitor 13 boarded wagon 0.
Visitor 5 boarded wagon 0.
Visitor 6 boarded wagon 0.
Visitor 16 boarded wagon 0.
Wagon 0 start going aroung the track.
Visitor 4 left wagon
Visitor 4 left the park, 3 visitor(s) remaining in the park.
Visitor 9 visited the park.
Visitor 9 wait for a ride.
Visitor 13 left wagon
Visitor 13 left the park, 3 visitor(s) remaining in the park.
Visitor 5 left wagon
Visitor 5 left the park, 3 visitor(s) remaining in the park.
Visitor 6 left wagon
Visitor 6 visited the park.
Visitor 6 wait for a ride.
Visitor 16 left wagon
Wagon 0 succesfully unloaded all passengers.
Visitor 9 boarded wagon 0.
Visitor 6 boarded wagon 0.
Visitor 16 visited the park.
Visitor 16 wait for a ride.
Visitor 16 boarded wagon 0.
Wagon 0 start going aroung the track.
Visitor 9 left wagon
Visitor 9 left the park, 2 visitor(s) remaining in the park.
Visitor 6 left wagon
Visitor 6 left the park, 2 visitor(s) remaining in the park.
Visitor 16 left wagon
Visitor 16 left the park, 2 visitor(s) remaining in the park.
Wagon 0 succesfully unloaded all passengers.
End Simulation.
```

OK !