```
// $Header: /home/cvs/t21617/samuel.riedo/s2/CooeeCooee.java,v 1.14 2017-05-01 09:02:46 samuel.riedo Exp $

import java.util.concurrent.locks.*;
import java.util.*;

/**
 * Simulation of chicks eating food hunted by two parents. Chicks and parents are
 * synchonized by monitor implemented as "Signal and Continue".
 *
 * @param string[]
 *                  1: Number of chicks process.
 *                  2: Chicks process iterations.
 *                  3: Nest maximal food capacity.
 *                  4: Parent hunting sucess rate (in %, 50 = 50%).
 */
public class CooeeCooee{

    private static int chicksInNest = 0;                                    // Number of chick in the nest.
    private static int chicksIterations = 53;                               // Number of chicks thread iterations.
    private static int nestFoodCapacity = 7;                                // Nest maximal food capacity.
    private static int huntingSuccesRate = 50;                              // Succes rate parent has when hunting (in %).
    private static volatile int nestFood = 0; NOOOO ! See page 3            // Nest food portions available for chicks.
    private static volatile int chicksNumber = 17;                          // Number of chicks process to be started.
    private static Parent parentOne;                                        // First parent thread.
    private static Parent parentTwo;                                        // Second parent thread.
    private static ReentrantLock lock = new ReentrantLock(true);            // Mutex functions.
    private static Condition full = lock.newCondition();                    // Signaled when nest not full of food.
    private static Condition empty = lock.newCondition();                   // Signaled when nest has no reamaining food.
    private static Thread[] chicksProcess;                                  // Table countening all Chicks process.
    private final static int PARENT_REST_TIME = 50;                         // Parent max rest time in nanoseconds.
    private final static int CHICK_SLEEP_TIME = 40;                         // Chick max sleep time in nanoseconds.

                                    These should be declared in the monitor, not here.

    /**
     * Program entry point. Create, start and launch all Chicks and Parents threads.
     * Read user parameters:
     * @param string[]
     *                  1: Number of chicks process.
     *                  2: Chicks process iterations.
     *                  3: Nest maximal food capacity.
     *                  4: Parent hunting sucess rate (in %, 50 = 50%).
     */
    public static void main(String[] args){
        int argsl = args.length;                                            // Main programm user arguments.
        switch (argsl) {
        case 4:
            huntingSuccesRate = Integer.valueOf(args[--argsl]);
        case 3:
            nestFoodCapacity = Integer.valueOf(args[--argsl]);
        case 2:
            chicksIterations = Integer.valueOf(args[--argsl]);
        case 1:
            chicksNumber = Integer.valueOf(args[--argsl]);
        }
        System.out.println("--Parameters--");
        System.out.printf("Chicks number: %d\n", chicksNumber);
        System.out.printf("Chicks iterations: %d\n", chicksIterations);
        System.out.printf("Max food capacity: %d\n", nestFoodCapacity);
        System.out.printf("Hunting sucess rate: %d\n\n", huntingSuccesRate);

        chicksProcess = new Thread[chicksNumber];

        createThreads();
        startThreads();

        try{
            for (int i = 0; i<chicksNumber; i++){
                chicksProcess[i].join();
            }
            parentOne.interrupt();
            parentTwo.interrupt();
            parentOne.join();
            parentTwo.join();
        }
        catch(InterruptedException e){
            System.out.println("Can't join on chick thread.");
            e.printStackTrace();
        }
        System.out.println("Simulation terminated.");

    }

    /**
     * Create "chicksNumber" Chick threads and two Parent threads.
     * Parent thread ID are 1 and 2.
     * Chicks thread ID start from 0 to chicksNumber.
     */
    private static void createThreads(){
        System.out.printf("Creating threads...\n");
        for (int i = 0; i < chicksNumber; i++) {
            chicksProcess[i] = new Chick(i);
        }
        parentOne = new Parent(1); // 1 = first parent ID
        parentTwo = new Parent(2); // 2 = second parent ID
    }

    /*
     * Start all Chick and Parent threads.
     * Chicks threads are started first after shuffle them.
     */
    private static void startThreads(){
        System.out.printf("Starting threads...\n\n");
        Collections.shuffle(Arrays.asList(chicksProcess));
        for (int i = 0; i < chicksNumber; i++) {
            chicksProcess[i].start();
        }
        parentOne.start();
```

```
            parentTwo.start();
        }

        /**
         * Simulate the behavior of a chick in a nest. A chick repeat chicksIterations times the following
         * tasks before leaving the nest:
         *   - sleep
         *   - get food
         *   - eat
         *   - digest
         * As digest is only a method that print a message, it hasn't been implemented to simplify
         * the program output.
         */
        static class Chick extends Thread{

            private int id;                                         // Chick thread unique ID.
            private int portionEaten=0;                             // Number of portion eaten by the chick.

            /**
             * Chick constructeur, set an ID to this thread.
             *  @param id - This thread ID.
             */
            public Chick (int id){
                this.id = id;
            }

            /**
             * Simulate the following tasks:
             *   - sleep
             *   - get food
             *   - eat
             *   - digest.
             * Do it chicksIterations times.
             */
            @Override
            public void run(){
                chicksInNest++;
                while(portionEaten<chicksIterations){
                    sleep();
                    eat();
                }
                chicksInNest--;  Aha !  this is a critical section, and therefore should be syncrhonised in the monitor !
                System.out.printf("Chick %d leave the nest, %d chick(s) remaining.\n", this.id, chicksInNest);
            }

            /**
             * Simulate a nap by setting the thread in a sleep state for CHICK_SLEEP_TIME nanoseconds.
             */
            public void sleep(){
                System.out.printf("Chick %d sleep.\n", this.id);
                try{
                    Thread.sleep(0,(int)(CHICK_SLEEP_TIME*Math.random()));
                }
                catch(InterruptedException e){
                    System.out.printf("Error while chick %d tried to sleep.\n", this.id);
                    e.printStackTrace();
                }
            }

            /**
             * Try to get food and eat it.
             * This method use the Monitor to resolve the following critical section problem:
             *    Only one chick or parent can modify nestFood value at the same time.
             */
            public void eat(){
                Monitor.getFood();
                this.portionEaten++;
                System.out.printf("Chick %d eat a portion, %d remaining in the nest.\n", this.id, nestFood);
            }
        }

        /**
         * Simulate the behavior of a bird hunting to feed his chicks. While there is chicks in the nest,
         * the two parents does the following tasks:
         *   - hunt
         *   - bring food back to the nest
         *   - take a rest.
         */
        static class Parent extends Thread{

            private int id;                                         // Parent thread unique ID.
            private int food=0;                                     // Hunted food portions.

            /**
             * Parent constructeur, set an ID to this thread.
             *  @param id - This thread ID.
             */
            public Parent(int id){
                this.id = id;
            }

            /**
             * Simulate the following tasks:
             *   - hunt
             *   - bring food back to the nest
             *   - take a rest.
             */
            @Override
            public void run(){
                do{
                    hunt();
                    bringBackFood();
                    rest();
                } while(chicksInNest>0);
                System.out.printf("Parent %d terminate\n", this.id);
```

```
        }

        /**
         * Simulate the process of hunting food.
         * The probability to bring nothing back is given by the variable huntingSuccesRate.
         * When the Parent succed to bring some food back, the number of portion is between
         * 1 and the variable nestFoodCapacity.
         */
        public void hunt(){
            System.out.printf("Parent %d start hunting.\n", this.id);
            if(Math.random()*100>huntingSuccesRate){
                this.food=(int)((Math.random()*(nestFoodCapacity-1))+1);
                System.out.printf("Parent %d hunted %d food portions\n", this.id, this.food);
            }
            else{
                System.out.printf("Unlucky parent %d didn't catch anything.\n", this.id);
                this.food=0;
            }
        }

        /**
         * Bring back hunted food to the nest.
         * This method use the Monitor to resolve the following critical section problem:
         *    Food can only be added to the nest when their is no remaining food. The variable
         *    nestFood can only be modified by one parent or chick at the same time.
         */
        public void bringBackFood(){
            Monitor.addFood(this.food);
            System.out.printf("Parent %d bring %d portion(s) of food.\n", this.id, this.food);
        }

        /**
         * Simulate a nap by setting the thread in a sleep state for PARENT_REST_TIME nanoseconds.
         */
        public void rest(){
            System.out.printf("Parent %d rest.\n", this.id);
            try{
                Thread.sleep(0,(int)(PARENT_REST_TIME*Math.random()));
            }
            catch(InterruptedException e){
                // If their is 0 chicksInNest, printf nothing as it's the end of simulation.
                if(chicksInNest>0){
                    System.out.println("Exception happend while parent was sleeping.");
                    e.printStackTrace();
                }
            }
        }
    }

    /*
     * Provide ressources to resolve critical sections problem.
     * The monitor is used by the following classes:
     * Parent: addFood method.
     * Chick: getFood method.
     */
    static class Monitor{
        /**
         * Update nestFood variable.
         * A parent can only add food to the nest when their is no
         * remaining food.
         *   @param portions - food portions to be added to the nest.
         */
        public static void addFood(int portions){
            lock.lock();
            try{
                while(nestFood>0){
                    empty.signal();
                    full.await();
                }
                nestFood=portions;
                empty.signal();
            }
            catch(InterruptedException e){
                // If their is 0 chicksInNest, printf nothing as it's the end of simulation.
                if(chicksInNest>0){
                    System.out.println("Exception happend while adding food to the nest.");
                    e.printStackTrace();
                }
            }
            finally{
                lock.unlock();
            }
        }

        /**
         * Update nestFood variable.
         * A check can only take one food portion at the same time.
         */
        public static void getFood(){
            lock.lock();
            try{
                while(nestFood==0)
                    empty.await();
                nestFood--;
                full.signal();
            }
            catch(InterruptedException e){
                System.out.println("Exception happend while getting food from the nest.");
                e.printStackTrace();
            }
            finally{
                lock.unlock();
            }
        }
    }
}
```

Since nestFood is NOT declared in the monitor, how can the monitor be sure its value is correct ?

Why the signal ? If nestFood is greater than 0, then there shouldn't be any chicks waiting !

By the way, the usage of full and empty doesn't correspond to the comments given for the variables. That's commandment #3

Bug: suppose that there are 3 chicks waiting. The parent deposits a quantity of 4 portions. The parent only wakes up 1 chick, and the others are left waiting. They should be allowed to eat right away. Hint: use signalAll()

```
/*
** $Log: CooeeCooee.java,v $
** Revision 1.14  2017-05-01 09:02:46  samuel.riedo
** Final version
** Changes:  - Spell check
**           - Indentation check
**           - Ten commandements check
**
** Revision 1.12  2017-05-01 08:13:24  samuel.riedo
** Bug corrected when using no default parameters:
**   Effect: The variable chicksProcess[] was initialized before
**           chicksNumber was assigned by user. This produced an
**           exception if the user change the first parameter
**           when starting the simulation.
**   Correction: chicksProcess[] is now initialized after
**               reading args[].
**
** Revision 1.11  2017-05-01 08:00:27  samuel.riedo
** Delete a wrong if statement in addFood method.
** If all chicks were sleeping and a parent tried to add food, the if condition would
** be wrong as their isn't any chick waiting for food.
** The food hunted by the parent would been throw away in that case.
**
** Revision 1.10  2017-05-01 07:40:34  samuel.riedo
** Change all Exception to InterruptedException as we only need to catch these.
** Add a test in several catch statements to do nothing is it's the end of the simulation, this
** is done because the main method interrupt parent threads when their is no more active chick
** in the nest, which my throw an exception is the thread is in the monitor or sleeping.
**
** Revision 1.9  2017-05-01 06:43:20  samuel.riedo
** At the end of the program, a parent can be stuck in the monitor when the
** number of food portion is greater than 0 and their is no remaining chick(s) to
** eat it.
** To correct this, a join on all chick threads was added in main method. After the
** join, the parent thread are interrupted (as their is no more chick, they are not
** relevant anymore.)
** Meanwhile, some minor change were made to respect the ten commandements.
**
** Revision 1.8  2017-04-29 13:58:16  samuel.riedo
** Added a join on all chicks threads in main method. This prevent the main method to
** finish before all threads are terminated and this is also used to interrupt a remaining
** runnin parent when all chicks have left the nest.
**
** Revision 1.7  2017-04-29 11:40:01  samuel.riedo
** Problem corrected:
**  - When all chicks have left the nest, the parents doesn't always terminate.
**  - To correct this, I added an extra condition in addFood(int portions) method (located in Monitor class).
** The condition verify if their is chick(s) waiting for food before blocking parent(s) if their is still
** remaining food in the nest.
**
** Revision 1.6  2017-04-29 10:58:25  samuel.riedo
** Chick implemented, starting now test.
**
** Revision 1.5  2017-04-29 10:40:06  samuel.riedo
** First version of the monitor implemented. Still not tested as Chick isn't implemented.
**
** Revision 1.4  2017-04-29 10:22:32  samuel.riedo
** Class parent implemented. The followings functions have been added:
**  - hunt
**  - bringBackFood
**  - rest
** None of them have been tested, as the monitor isn't implemented.
**
** Revision 1.3  2017-04-29 08:46:13  samuel.riedo
** Add CVS header
**
** Revision 1.2  2017-04-29 08:39:52  samuel.riedo
** Add programm skeleton. Create Parent and Chick class. Add first constants.
**
** Revision 1.1  2017-04-03 07:56:42  samuel.riedo
** Initial commit to test
**
*/
```

```
java CooeeCooee
--Parameters--
Chicks number: 17
Chicks iterations: 53
Max food capacity: 7
Hunting sucess rate: 50

Creating threads...
Starting threads...

Chick 1 sleep.
Chick 6 sleep.
Chick 7 sleep.
Chick 15 sleep.
Chick 4 sleep.
Chick 9 sleep.
Chick 13 sleep.
Chick 8 sleep.
Chick 10 sleep.
Chick 5 sleep.
Parent 2 start hunting.
Parent 1 start hunting.
Chick 2 sleep.
Chick 12 sleep.
Chick 16 sleep.
Chick 3 sleep.
Chick 11 sleep.
Chick 14 sleep.
Chick 0 sleep.
Unlucky parent 1 didn't catch anything.
Parent 2 hunted 2 food portions
Parent 1 bring 0 portion(s) of food.
Chick 15 eat a portion, 0 remaining in the nest.
Chick 15 sleep.
Chick 10 eat a portion, 0 remaining in the nest.
Chick 10 sleep.
Parent 2 bring 2 portion(s) of food.
Parent 2 rest.
Parent 1 rest.
Parent 2 start hunting.
Parent 1 start hunting.
Unlucky parent 1 didn't catch anything.
Parent 1 bring 0 portion(s) of food.
Parent 1 rest.
Parent 2 hunted 1 food portions
Parent 2 bring 1 portion(s) of food.
Parent 2 rest.
Chick 1 eat a portion, 0 remaining in the nest.
Chick 1 sleep.
Parent 1 start hunting.
Unlucky parent 1 didn't catch anything.
Parent 1 bring 0 portion(s) of food.
Parent 1 rest.
Parent 2 start hunting.
Parent 2 hunted 6 food portions
Parent 2 bring 6 portion(s) of food.
Parent 2 rest.
Chick 9 eat a portion, 5 remaining in the nest.
Chick 9 sleep.
Parent 1 start hunting.
Parent 1 hunted 4 food portions
Chick 13 eat a portion, 4 remaining in the nest.
Chick 13 sleep.
Chick 2 eat a portion, 0 remaining in the nest.
Parent 2 start hunting.
########################################..... 3746 lines skipped ......
########################################Parent 2 rest.
Chick 5 eat a portion, 3 remaining in the nest.
Chick 5 sleep.
Chick 7 eat a portion, 2 remaining in the nest.
Chick 7 sleep.
Parent 2 start hunting.
Parent 2 hunted 1 food portions
Chick 5 eat a portion, 1 remaining in the nest.
Chick 5 sleep.
Parent 1 bring 5 portion(s) of food.
Parent 1 rest.
Chick 7 eat a portion, 0 remaining in the nest.
Chick 7 leave the nest, 1 chick(s) remaining.
Chick 5 eat a portion, 4 remaining in the nest.
Chick 5 leave the nest, 0 chick(s) remaining.
Chick 9 leave the nest, 3 chick(s) remaining.
Chick 7 eat a portion, 2 remaining in the nest.
Chick 7 sleep.
Parent 1 start hunting.
Parent 1 hunted 5 food portions
Chick 15 eat a portion, 1 remaining in the nest.
Chick 15 leave the nest, 2 chick(s) remaining.
Chick 5 eat a portion, 0 remaining in the nest.
Chick 5 sleep.
Parent 2 bring 0 portion(s) of food.
Parent 2 rest.
Parent 2 start hunting.
Unlucky parent 2 didn't catch anything.
Parent 2 bring 0 portion(s) of food.
Parent 2 rest.
Parent 2 start hunting.
Parent 2 hunted 4 food portions
Parent 2 bring 4 portion(s) of food.
```

4 + .25 = 4.25