Homework Assignment

Concurrent and Distributed Systems

Student: Julia Florea
Calculatoare Engleza
Anul III, Grupa 3.2 A
Facultatea de Calculatoare, Automatică și Electronică

1 Introduction

1.1 Problem statement

Uncle John heard that selling organic eggs has become a great business nowadays. Therefore, he decided to breed hens and create some farms for producing organic eggs. The hens lay eggs which are picked up by his employees and sent to the farm headquarter. Uncle John relies on you to help him implement the farm plan by using concurrency in Java.

1.2 Instructions

- **There are multiple farms:**
  -The hens are raised in farms. The number of farms is a random number between 2 and 5;
  -Each farm is designed as a matrix with N x N where $100 \leq N \leq 500$;
  -Each farm has a list of hens;
  -Every few seconds the farm monitoring system will request the position of the hens inside the farm;
  -There cannot be more than N / 2 hens for a farm.

- **There are hens which produce eggs:**
  -The hens are spawning randomly in each farm at a random place, at a random time t, where $500 \leq t \leq 1000$ (milliseconds);
  -Two hens cannot be in the same place;

- **The hens lay eggs by:**
  -Moving in one direction (left, right, up, down);
  -When they reach the fence of the farm, they move in any other direction than the fence;
  -With each move, they lay an egg;
  -If the hen is surrounded by other hens and cannot move in any direction, it stops for a random time t, where $10 \leq t \leq 50$ (milliseconds);
  -Each time an egg is produced, the farm monitoring system informs the employees about the position where the egg is located;
  -After an egg is created, the hen needs to rest until it is able to produce another egg (resting time is 30 milliseconds);
  -The hens will produce eggs while it is alive.

- **There are also farm employees:**
  -The employees await to receive eggs from the farm;
  -They can always read information from the farm monitoring system regarding the number of laid eggs, but they won't be allowed access when the sensors notify the monitoring system that an egg was laid or when the

farm monitoring system itself asks the sensors about the new laid eggs;
-A maximum of 10 farm employees can read from the monitoring system at a time;
-A random time must pass between two consecutive monitoring system readings;
-All employees will deliver the eggs to the farm headquarter;
-The employees' number is known from the beginning. There are more than 8.

- **The farm headquarter will do the following:**
  -The headquarter contains all the farms;
  -It creates the farms;
  -It creates the hens and assigns them to random farms (remember that each hen needs to register by itself to the farm).

1.3 Requirements and objectives

The project aims at developing a software for experimenting with concurrency in Java.The (concurrent) control flow in the project is rather subtle.The moving of hens and eggs report is initiated independently by each hen, as each of them is running as separate thread of control. Because of that, different hens may execute the move action and the egg report concurrently. But calling report also triggers calling the individual report method of all the hens registered within the farm. Both report and move methods are synchronized, which means that they will not be called concurrently for a given object.

2 Project perspective

Concurrency is the ability to run several or multi programs or applications in parallel. The backbone of Java concurrency is represented by threads (a lightweight process, which has its own files, stacks and can access the shared data from other threads in the same process). The throughput and the interactivity of the program can be improved by performing time-consuming tasks asynchronously or in parallel.
Java is a multi-threaded programming language, which means we can develop multi-threaded programs using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time, making optimal use of the available resources, especially when the computer has multiple CPUs.
A concurrent program is composed of more sequential programs that can be executed in parallel. Sequential programs of a concurrent program are also called processes .

2.1 Project approach

The proposed problem was solved by using concurrent concepts in Java such as semaphores, monitors and locks, threads for implementing the hens and the farm's management system, synchronized methods etc.

3 Project architecture

The project consists of the following classes : **Main**, **Farm**, **Egg**, **Hen**, **Employee**, **FarmHeadquarters**.

3.1 Main Class

This class is meant for testing the program and creating an object of the FarmHeadquarters class .

```
1  package eggs_farm;
2
3  public class Main {
4
5
6      public static void main(String[] args) {
7
8
9          FarmHeadquarters F = new FarmHeadquarters();
10         F.start();
11
12     }
13 }
14
```

Figure 1: Main class

3.2 Farm Class

This class manages the hens, stores the eggs and notifies the farm headquarters whenever the eggs produced by the hens are enough for a new stock.
Each farm is created as a matrix of N x N and the number of farms is a random one.
At a random time, the farm's managing system requests the exact location of each hen, which is done by using coordinates for the positions.
Due to the fact that the class must fulfill many requirements simultaneously, it is created as a thread, in order to be executed independently. When an instance of this class is created, it must be assigned to it a random size and a random number of eggs to produce.
For making sure that the farm works properly, it is created another thread, which requests the position of each hen at a random time. The method run() verifies at the beginning the number of eggs stored at that moment in the farm. When a certain number of eggs is reached, the farm notifies the headquarters that the eggs are ready to be delivered. Then, the headquarters notifies the employees, who will transport the eggs.
In order to be sure that maximum 10 employees can transfer the eggs from the farm to the headquarters, it is created a semaphore. Before transferring an egg, the employees request permission and then they release it. A methos for moving the hens in 4 possible directions(up, down, left, right), in a random manner, is also implemented.

```
7  public class Farm extends Thread {
8
9      ReentrantLock move_lock = new ReentrantLock(true);
10     FarmHeadquarters FarmHeadquarters = null;
11     Vector<Hen> hens = new Vector<Hen>();
12     Vector<Egg> eggs = new Vector<Egg>();
13     Semaphore semaphore = new Semaphore(10, true);
14     int grid_size = 0;
15     int[][] grid; // the farm is a matrix N x N of random size between 100 and 500
16     int code; // the code/ number for identifying a certain farm
17     int no_eggs_created = 0;
18     int no_egg_needed;
19
20     public Farm(FarmHeadquarters farmHeadquarters, int code) {
21         this.FarmHeadquarters = FarmHeadquarters;
22         grid_size = (int) (Math.random() * (500 - 100) + 100);
23         grid = new int[grid_size][grid_size];
24         this.code = code;
25         no_egg_needed = (int) (Math.random() * (100 - 25) + 25);
26         // number of eggs needed in each farm
27         new Thread(() -> { // a new thread for handling the position request
28             RequestPositions();
29         }).start();
30
31         this.start();
32     }
33
34     public void run() {
35         while (FarmRunning()) { // announces if there are enough eggs to be transported
36             if (eggs.size() > 9) {
37                 FarmHeadquarters.announce(this);
38             }
39
40         }
41         stopHens(); // enough eggs were produced, so the hens can rest
42         while (!eggs.isEmpty()) { // while there are enough eggs for a transport, we announce the farm
43                                   // headquarters and we can empty the eggs list
44             FarmHeadquarters.announce(this);
```

Figure 2: Farm class

3.3 FarmHeadquarters Class

FarmHeadquarters represents the most important part of the simulation, as all the other classes rely on it in order to function properly. It contains all the

farms, creates them, as well as creating the hens and assigning them to random farms.
For an object of this type, the constructor creates farms, opens the TCP/IP communication line and then also creates the employees. All the entities previously created are stored in a vector. For increasing the performance, it is used an ArrayList, but this requires an additional synchronization. At the same time, the methods for the vectors are thread safe.
This class has also been implemented using a thread. The run() method assigns each hen to a farm after a random amount of time.
The farm headquarters adds the hens and maintains the TCP/IP line open until all the farms notify that they have finished the number of eggs allocated.
When a farm finishes its production, the headquarters is notified. Furthermore, when all the farms finish their job, all the employees stop working.

```
public class FarmHeadquarters extends Thread {

    // we create the lists of farms, employees and eggs
    Vector<Farm> farms = new Vector<Farm>();
    Vector<Employee> employees = new Vector<Employee>();
    Vector<Egg> eggs = new Vector<Egg>();
    ServerSocket socket = null;
    int active_farms; // the farms which produce eggs
    static int no_hens = 0;

    FarmHeadquarters() {
        active_farms = 3; // we create the farms(a random number between 2 and 5)
        for (int i = 0; i < 3; i++) {
            farms.add(new Farm(this, i));
        }
        openTransportLine();

        // the maximum number of employees
        for (int i = 0; i < 10; i++) {
            employees.add(new Employee(this, farms));
            System.out.println("The employee" + i + " was called for duty.");
        }
        System.out.println("The case is running.");
    }

    public void run() {
        while (farmsRunning()) {
            addHens(); // we add the hens at a random moment of time, while the farms are active and
                       // working

            try {
                Thread.sleep((int) (Math.random() * (1000 - 500) + 500));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("All farms have finished their work, having a total of " + eggs.size()
                + " eggs, which have been delivered to the farm headquarters. ");
        try {
```

Figure 3: FarmHeadquarters class

3.4 Egg Class

The main objective of this class is testing. It contains two fields : serial and hen serial. Each farm and each hen has a serial code, allocated uniquely, which is used for debugging and testing. The class must implement the Serializable interface, without which it could not be transferred through TCP/IP communication.

```java
package eggs_farm;

import java.io.Serializable;

public class Egg implements Serializable {
    int serial; // serial number for egg
    int Hen_serial; // serial number for hen

    public Egg(int Hen_serial, int serial) {
        this.Hen_serial = Hen_serial; // initialising the parameters of the constructor
        this.serial = serial;
    }

}
```

Figure 4: Egg class

3.5 Hen Class

This class is a significant one. When a hen is created, a code is assigned to it and sent to a farm. When it arrives to the farm, it assigns to it a random position on the grid. It is executed in a while loop, which checks if the flag for the necessary eggs is true.If it is, then an egg is produced.
For producing an egg, a hen must move one step in any direction (up, down, left, right) which is available. When reaching a fence, the hen must go back. When initializing the objects, for each of them it is created an array composed of 4 numbers from 0 to 3, each representing a direction.
When producing the eggs, the first step is to randomize the array, so as the direction to be chosen randomly.
Then, the hen must find out whether there is enough space for storing the eggs. If there is, then the farm marks that position as busy. After that, the hen changes its position and creates a new egg.
If the egg has been created successfully, the hen gives the egg to the farm, then waits 30*X milliseconds. If an egg has not been created, the hen waits from 10 to 50 milliseconds. When the waiting time is over, the process is repeated.

```java
public class Hen extends Thread {
    int x; // x and y compose the coordinates for the position of the eggs based on the
           // direction chosen by the hen
    int y;
    private Farm Farm = null;
    boolean necessaryEggs;
    private final int serial; // serial number of the hen
    int[] MoveDirection = null; // matrix with 4 directions (left, right, up, down) having a random order

    Hen(int serial) {
        this.serial = serial;
        necessaryEggs = true;
        MoveDirection = new int[4];
        MoveDirection[0] = 0;
        MoveDirection[1] = 1;
        MoveDirection[2] = 2;
        MoveDirection[3] = 3;
    }

    public void Assign(Farm Farm, int x, int y) { // function to assign hens to the farms in a random place
        this.Farm = Farm;
        this.x = x;
        this.y = y;
    }

    public void run() {
        boolean moved;
        while (necessaryEggs) {
            moved = tryToMove(); // while there are necessary eggs for the farm, the hen tries to move in a
                                 // certain direction
            if (moved) {
                Egg egg = new Egg(Farm.getCode(), this.serial);
                Farm.TransferEgg(egg); // if the hen was able to move and it laid an egg, then we create the egg and
                                       // transfer it to the farm
```

Figure 5: Hen class

3.6 Employee Class

The Employee class waits to receive eggs from the farm. They can read information from the farm's managing system about the number of eggs, but the maximum number of employees that can read at the same time is 10.
The employees are created at the same time as the farm headquarters and they are also represented by a thread. The run() method checks if the headquarters is still running.
As long as the employees are running, the information from the farms is being read continuously. Then, it is verified if the employees have been assigned to a farm for performing the transportation of the eggs.
In order to assign a farm to an employee, the headquarters is constantly used. When no farm is assigned, the employee takes the first farm chosen by the headquarters.

```
 9  public class Employee extends Thread {
10
11      FarmHeadquarters FarmHeadquarters = null;
12      private Vector<Farm> farms = null;
13      private volatile Farm source_Farm = null; // the source farm is declared volatile in order to be sure that the
14                                                // variable is up to date and thread safe
15      Socket socket = null;
16
17      public Employee(FarmHeadquarters FarmHeadquarters, Vector<Farm> farms) {
18          this.FarmHeadquarters = FarmHeadquarters;
19          this.farms = farms; // this class must have access to the farms in order to be able to read the data
20          this.start();
21      }
22
23      public void run() {
24          while (FarmHeadquarters.farmsRunning()) {
25              readData(); // reads the information from the monitoring system of the farm headquarters
26              if (getSourceFarm() != null) { // when a farm is assigned to retrieve an egg, it completes the task
27                  transportEggs();
28              }
29              setSource(null); // resets the source_Farm
30          }
31      }
32
33      private void transportEggs() {
34          // function to transport the eggs to the farm
35          if (source_Farm.RequestTransportAccess()) { // the source farm requests the access to the system
36              source_Farm.RequestMove(); // is locked
37              Vector<Egg> V = source_Farm.getEggList();
38              // the source farm receives the list of the eggs
39              if (V.isEmpty()) {
40                  // if there are no more eggs, release the lock
41                  source_Farm.MoveFinished();
42                  source_Farm.TransportFinished();
43                  return;
44              }
45              Egg egg = V.remove(0); // places the first egg in the list; releases the lock and permits access
46              source_Farm.MoveFinished();
47              source_Farm.TransportFinished();
```

Figure 6: Employee class

4 Application outline

4.1 Methods included in each class

- **Main Class:**

  - **main(String[] args)** - creates an instance of the FarmHeadquarters class and starts the thread, in order to test the whole program

- **Farm Class:**

- **Farm(FarmHeadquarters farmHeadquarters, int code)** - the constructor of the class, which takes the parameters farmHeadquarters (object of the FarmHeadquarters class) and code (the number for identifying a farm) and initializes them; we also declare and initialize the grid (the farm is a matrix N x N of random size between 100 and 500), the size of the grid, the number of necessary eggs; then, we create a thread for handling the position request;

- **run()** - announces if there are enough eggs to be transported; if the number of eggs is greater than 9, then we can announce the farm headquarters and stop the production and the hen can rest, else, we check if the list of eggs is empty and announce that a certain farm has stopped the production;

- **addHen(Hen production-hen)** - adds a hen from the farm headquarters on our grid (farm); x and y are the coordinates for the position of the hen; we implement a lock on the hen until it is assigned to a farm and generate random positions for each hen;

- **RequestMove()** - requests to move the hen in a direction (up, down, left, right); the hen can move randomly, but it has to avoid the fence;

- **RequestMoveDirection(int x, int y, int i)** - checks if the direction which the hen chooses is available or not; the function is not synchronized because the hen locks the lock before calling it; this is done in order to make sure that a hen which wants to move can check each possible direction before deciding that it can't move a certain way; if the function had been synchronized, more hens would have been able to enter the moving stage without actually being able to move; if their first choice had not been available, the hens would have been blocked;

- **MoveFinished()** - unlocks the lock when after the hens were moved;

-**TransferEgg(Egg egg)** - transfers the eggs to the farms; takes as parameter an instance of the Egg class; it locks the lock, adds the eggs to the list and increments the number of eggs created, then unlocks the lock implemented;

-**RequestPositions()** - while the farm is running, it requests the positions of the hens;

- **FarmRunning()** - checks if the farm is working properly;

- **needsHens()** - checks if the farm needs more hens; checks if the number of hens is less than N/2 and if the farm needs more eggs;

- **getCode()** - returns the code for identifying the farm;

- **getNoEggs()** - function used by the employees in order to find out the number of eggs;

- **getEggList()** - function used by the employees in order to see the list of the eggs;

- **stopHens()** - function to stop the hens from producing eggs;

- **RequestTransportAccess()** - function used by the employees to find out the information regarding the transport of the eggs;

- **TransportFinished()** - realeses the semaphore, announcing that the transport has been finished;

- **Hen Class:**

- **Hen(int serial)** - the constructor which initializes the serial number of the hens, sets the necessary eggs flag to true, creates the matrix for the moving directions of the hens;

- **Assign(Farm Farm, int x, int y)** - assigns hens to the farms at a random position;

- **run()** - while there are necessary eggs for the farm, the hen tries to move in a certain direction; if the hen was able to move and it laid an egg, then we create the egg and transfer it to the farm; after laying an egg, the hen has to rest before laying another one; if the egg was produced, then the hen can go ahead and produce another one as long as it is alive; if the hen is surrounded by other hens and can't move in any direction, so it waits for a random time until it can move again;

-**tryToMove()** - randomizes the directions of the matrix where hens can move; there exists a lock, so in order to make a move, we have to send a request; the hen tries to move in the 4 possible directions; when the move is finished, we can unlock the lock;

- **Move(int i)** - update the coordinates for the position of the egg based on the direction which the hen chooses;

- **GetPosition()** - returns the position of the eggs; the leftmost numbers are represented by x and the rightmost ones by y;

- **stopProduction()** - updates the necessaryEggs variable to false, in order to stop the production, because eggs are no longer needed;

- **randomizeDirections()** - randomizes the directions of the matrix for moving the hens;

- **Egg Class:**

-**Egg(int Hen-serial, int serial)** - the constructor which initializes the serial n umbers for the hens and the eggs;

- **Employee Class:**

- **Employee(FarmHeadquarters FarmHeadquarters, Vector¡Farm¿ farms)** - the constructor; it must have access to all farms in order to be able to read the data needed;

-**run()** - reads the information from the monitoring system of the farm headquarters; when a farm is assigned to retrieve an egg, it completes the task, then resets the source farm;

- **transportEggs()** - function to transport the eggs to the farm; the source farm requests the access to the system, then it receives the list of the eggs; if there are no more eggs, release the lock; then, it places the first egg in the list, releases the lock and permits access;

-**readData()** - function to read the information from the farm monitoring system and wait after each reading for a random time;

- announce(Farm F) - function to announce the farm if an egg does not have a source of transport and assign one to it;
- **setSource(Farm F)**- synchronized function to set the source to the farm; it is synchronized because it can be called by any case through announce or through Hen in a loop;

- **getSourceFarm()** - returns the source farm of an egg;m

- **FarmHeadquarters Class:**

- **FarmHeadquarters()** - constructor which creates a random number of farms;

- **run()** - we add the hens at a random moment of time, while the farms are active and working;

- **openTransportLine()**- function to create new thread for the transport line of the eggs;

- **accepteggs()** - function to accept the eggs received by the farms, which were transported by the employees; makes a connection through a socket in order to transport the eggs; if a connection had been made,the socket closes;

- **addHens()** - adds the hens to the farm;

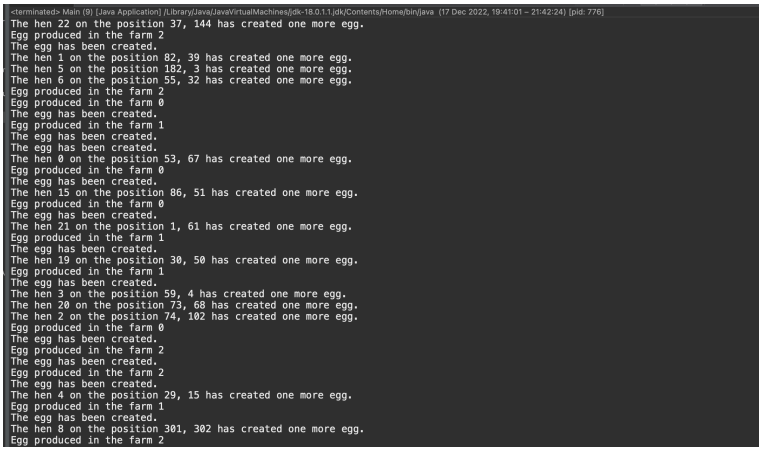
- **announce(Farm F)** - function to announce the employees that the a certain farm has eggs ready to be transported;

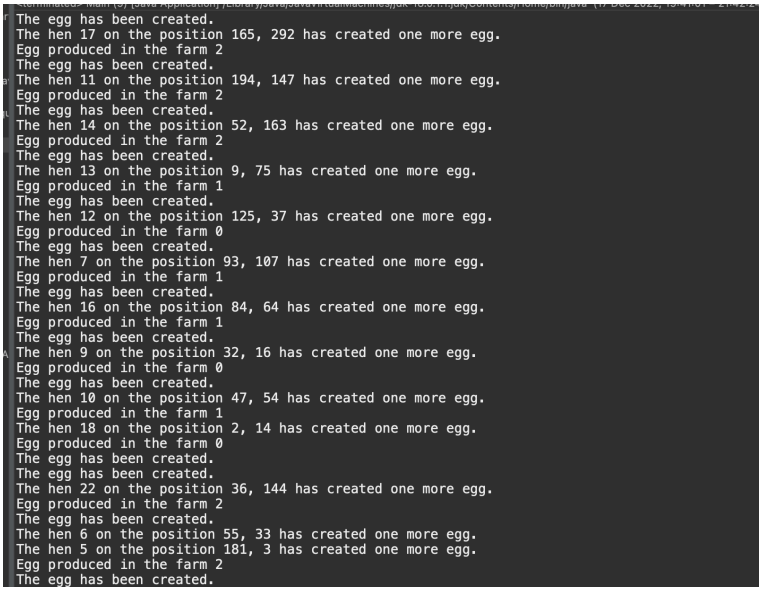
- **announceStop(Farm Farm)** - function called by the farms which stop the production of the eggs;

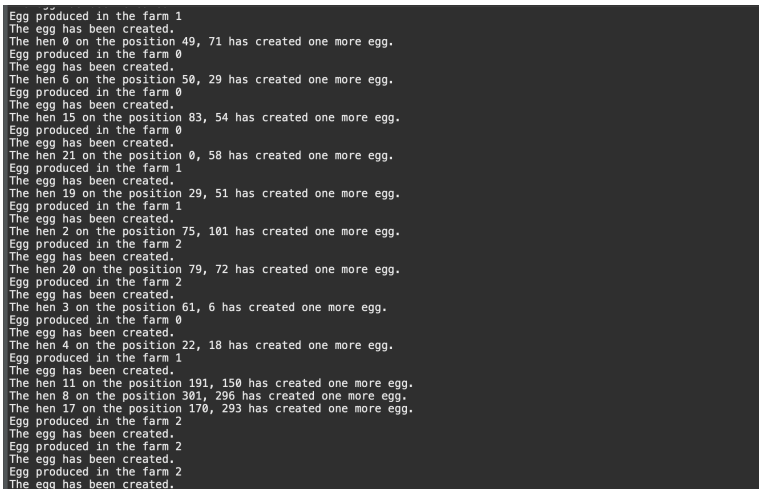
-**farmsRunning()** - function to check if there are farms which are working (producing eggs);

5 Experiments and results

By running the algorithms of each method described above, the program is expected to display the positions of the hens, to announce when an egg is produced, if there are needed more eggs or if the production has stopped and the assignation of the eggs to the farms.
Let us consider a few examples with their corresponding outputs below.


<terminated> Main (9) [Java Application] /Library/Java/JavaVirtualMachines/jdk-18.0.1.1.jdk/Contents/Home/bin/java (17 Dec 2022, 19:41:01 – 21:42:24) [pid: 776]
The hen 22 on the position 37, 144 has created one more egg.
Egg produced in the farm 2
The egg has been created.
The hen 1 on the position 82, 39 has created one more egg.
The hen 5 on the position 182, 3 has created one more egg.
The hen 6 on the position 55, 32 has created one more egg.
Egg produced in the farm 2
Egg produced in the farm 0
The egg has been created.
Egg produced in the farm 1
The egg has been created.
The egg has been created.
The hen 0 on the position 53, 67 has created one more egg.
Egg produced in the farm 0
The egg has been created.
The hen 15 on the position 86, 51 has created one more egg.
Egg produced in the farm 0
The egg has been created.
The hen 21 on the position 1, 61 has created one more egg.
Egg produced in the farm 1
The egg has been created.
The hen 19 on the position 30, 50 has created one more egg.
Egg produced in the farm 1
The egg has been created.
The hen 3 on the position 59, 4 has created one more egg.
The hen 20 on the position 73, 68 has created one more egg.
The hen 2 on the position 74, 102 has created one more egg.
Egg produced in the farm 0
The egg has been created.
Egg produced in the farm 2
The egg has been created.
Egg produced in the farm 2
The egg has been created.
The hen 4 on the position 29, 15 has created one more egg.
Egg produced in the farm 1
The egg has been created.
The hen 8 on the position 301, 302 has created one more egg.
Egg produced in the farm 2


The egg has been created.
The hen 17 on the position 165, 292 has created one more egg.
Egg produced in the farm 2
The egg has been created.
The hen 11 on the position 194, 147 has created one more egg.
Egg produced in the farm 2
The egg has been created.
The hen 14 on the position 52, 163 has created one more egg.
Egg produced in the farm 2
The egg has been created.
The hen 13 on the position 9, 75 has created one more egg.
Egg produced in the farm 1
The egg has been created.
The hen 12 on the position 125, 37 has created one more egg.
Egg produced in the farm 0
The egg has been created.
The hen 7 on the position 93, 107 has created one more egg.
Egg produced in the farm 1
The egg has been created.
The hen 16 on the position 84, 64 has created one more egg.
Egg produced in the farm 1
The egg has been created.
The hen 9 on the position 32, 16 has created one more egg.
Egg produced in the farm 0
The egg has been created.
The hen 10 on the position 47, 54 has created one more egg.
Egg produced in the farm 1
The hen 18 on the position 2, 14 has created one more egg.
Egg produced in the farm 0
The egg has been created.
The egg has been created.
The hen 22 on the position 36, 144 has created one more egg.
Egg produced in the farm 2
The egg has been created.
The hen 6 on the position 55, 33 has created one more egg.
The hen 5 on the position 181, 3 has created one more egg.
Egg produced in the farm 2
The egg has been created.



6 Conclusions

To conclude, in developing a solution to a problem, it can be quite beneficial to consider the data and the format in which they are presented. It can be a waste of resources (time and memory in the case of computing) to manipulate data that communicates more information than needed to answer a given question.
The main purposes of this project are to implement the farm with its given instructions, making use of the prerequisites of concurrency in Java programming language . After the completion of the assignment, I became familiar with a various number of features provided by the concurrent interface (such as semaphores, locks, sockets, synchronized methods etc) and LaTeX. At the same time, I've managed to create a programming work style, which enables me to arrange the source code in a much more readable and organized way.
The most challenging part of the project was to move one hen at a time, as changing a hen's position / an egg's position or an employee who transfers an egg to a farm are actions that cannot be performed simultaneously. For making the right choice, I started by thinking of all the possible solutions and writing a basic approach for each one, until I have found the most efficient one. Therefore, I have implemented the moveLock() method.
Thus, the structure of the project is composed in such a way as we can make sure that an upper object does not terminate, except the case in which all the other objects controlled by it are already terminated.

7 References

[1] Classroom courses and laboratories

[2] LATEXproject site, http://latex-project.org/, accessed in April 2013.

[3] Geeks for geeks site, https://www.geeksforgeeks.org/

[4] Stack overflow site, https://stackoverflow.com/