Baciu Lidia-Ana-Maria
CTI - English Section <CE 3.1.A>

Santa Claus Workshop Plan

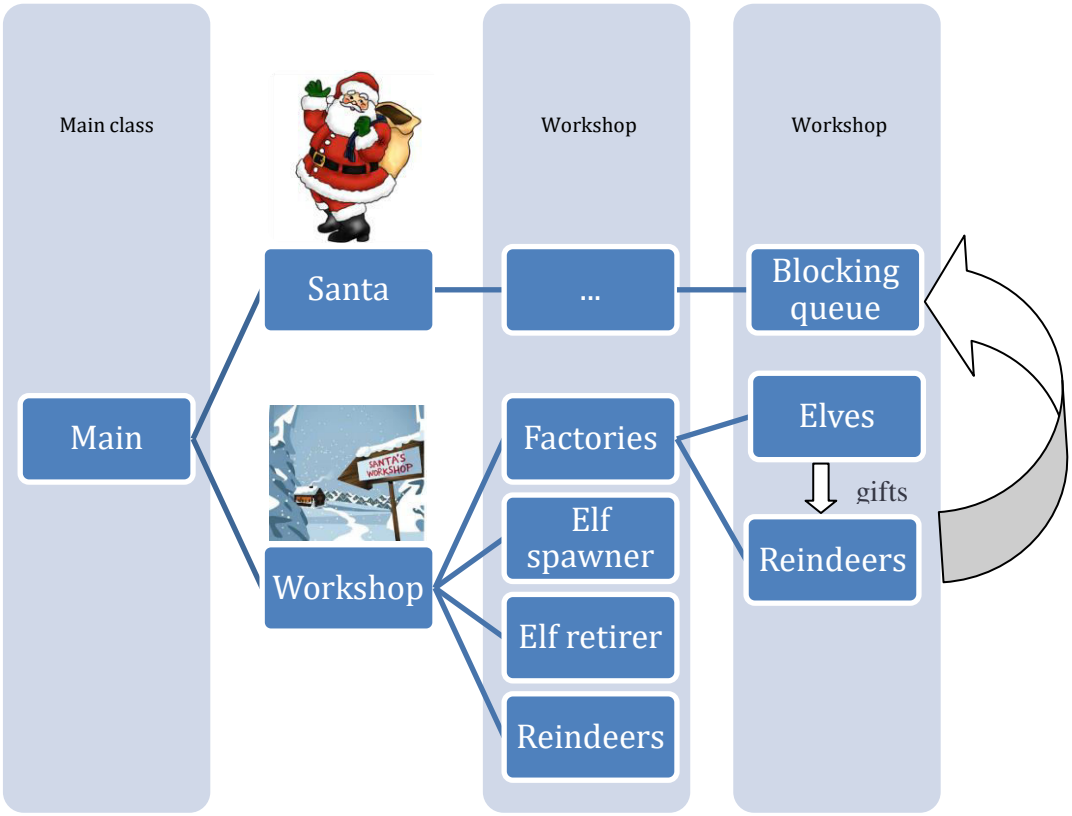
21st December 2018

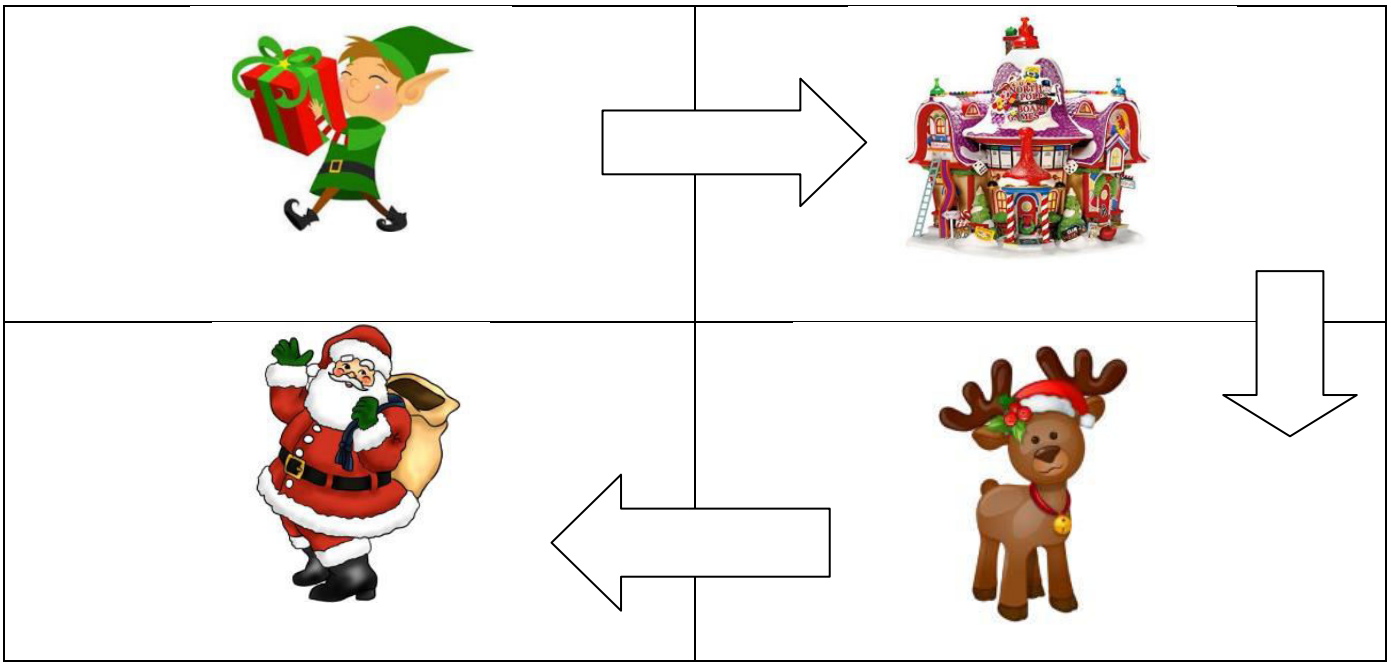
OVERVIEW

The problem is that Santa Claus wants his workshop to be the most efficient. He wants each elf to work independently and the reindeers to send him the gifts from the factories.

GOALS

1. Elves work independently and they create the maximum number of gifts. This means that when they hit a wall, they find out immediately where to move next, so they don't lose a gift.
2. Once a gift is created, they send it to the factory and the factory will ask all its elves where they are positioned.
3. If the elves are interrogated, the reindeers cannot enter the factory.
4. Retire an elf.
5. Give an elf time to sleep if he's in the diagonal area of the factory.
6. Send the gifts to Santa.


Main class
Workshop
Workshop
Main
Santa
Workshop
...
Factories
Elf spawner
Elf retirer
Reindeers
Blocking queue
Elves
gifts
Reindeers



TECHNICAL REPORT

My project has **9** classes (I have attached an UML diagram. (I don't really know how to create an UML diagram...but it seemed fine to me):

Main (which could also be the same with the Workshop class), Workshop, Factory, Elf, ElfSpawner, ElfRetirer and Reindeer, LockBasedQueue, SantaClaus.

1. *Workshop:* contains a random number of factories, the maximum number of elves (this results from the (sum of the dimension of each factory) /2, a well-known number of reindeers. Here I create the factories, I start the ElfSpawner and ElfRetirer and then I start the Reindeers.
2. *Factory*: This class uses Semaphores and synchronized and contains the following methods:
    a. aquireReindeersLock(int), releaseReindeersLock(int): synchronization on the list of gifts
    b. aquireElvesLock(int,String): this method also adds the current gift to the list and also calls the update method, in which the factory asks all its elves about their current positions
    c. releaseElvesLock()
    d. registerElf(Elf): it adds an elf to the list of elves. Each factory has its own list of registered elves
    e. registerReindeer(Reindeer): here is a little catch, as I assign from the start to each reindeer where it should go…
    f. getUpdates(): updates the matrix of the factory. This matrix is synchronized
    g. retire(Elf): it stops its loop and it also removes the elf from the list
    h. printNumberOfGifts(): I have used this method to know if the elf moves correctly and if it creates the maximum number of gifts. After all elves have finished their job (it's 25th of December or they were all retired), I get to see the statistics of each elf.
3. *ElfSpawner*: generates a random number, which represents the ID of the factory. If that factory has already a maximum number of elves, then it generates another id. Then it assigns the elf to the generated factory and starts its activity.
4. *Elf:* I did not use any synchronization in this class; it is a thread. It has the following methods:
    a. register(): when the elf got into the factory, the first thing it should do is to register (Factory method d.)
    b. getPositions(): returns the current and the last positions of the elf.
    c. retireElf(): it stops the loop from the run() method
    d. move(): this method is fragmented in 5 sub-methods:
        i. doNotHitWall(int): when the elf got into a wall, it has to move in another direction - fast! so it can generate a gift.
        ii. moveLeft(), moveRight(), moveUp(), moveDown(): each of these methods updates the current coordinates and the last ones. I need to know which were the last coordinates so I can free that position in the matrix

iii. For each of the above methods I check if my elf runs into another elf. If this happens, then my elf stays on this current position. It's worth mentioning that if this happens 4 times, my elf sleeps.

5. *ElfRetirer*: this is a thread. It retires an elf with a random id. [see Elf c. & Factory g.]
6. *Reindeer*: this is a thread. If the factory is available, then the reindeer enters and gets the gifts.
7. *SantaClaus:* this is a thread that starts in the main class, before the workshop is created. It shares a *LockBasedQueue* with the reindeers. It runs forever.
8. *LockBasedQueue*: this class implements two main methods: deque and enque. Basically, the reindeers "enque" an element, until the list becomes of the size 1000. Santa's job is to read from the list, so to "deque" the elements.

THINKING THE LOCKS

As I am not mastering the skill of understanding completely the synchronizations in Java, I will try to explain as better as I can how I thought about the solution for Santa.

What I have synchronized:

- **In the Factory class** I have **tried** to synchronized everything. Even though I understood that there is some also needed in the Elf class…
- I have 3 semaphores:
    - private Semaphore semElf = new Semaphore(1);
        - Is used when an elf tries to do something, e.g.: register (because it has to access the list of elves), retire (it also modifies the list of elves).
        - While getting the updates of the positions, it has to access the list of elves also.
    - private Semaphore listSemaphore = new Semaphore(1);
        - Is used when either the elf or the reindeer tries to add or to remove from the list of gifts. (the relationship producer-consumer between elf-reindeer)
    - private Semaphore semReindeer = new Semaphore(1);
        - Is used when an reindeer tries to access the list of gifts of each reindeer.

For the extra task 3

- I have kept only the following classes: Workshop, Factory, Elf, ElfSpawner.
- In the Workshop class I have created a cyclic barrier who works like this:
    - I set as the number of parties the maximum number of elves (this is known after the factories are created)
    - The barrier is passed as a parameter to the elf spawner and then to each of the elves.
    - I have translated this "finds itself in the diagonal area of the world (i.e. where x is very close to y)," as: `if(xCurrentPosition - yCurrentPosition < 2 && yCurrentPosition - xCurrentPosition < 2)`

- When an elf finds itself on the *principal diagonal* of the matrix, it will wait for the other elves to reach the barrier (meaning to be on the diagonal of the matrix also)

- For the sake of my time (and yours also) I have simplified the problem:
  - 1 factory with the dimension 10x10
  - 5 elves
  - No retirement and no reindeers.
- The output is:
  - 1 factories
  - Factory 0 has the dim 10
  - elf no.0 coordinates: 4,7
  - factory no.0has elf[id: 0]
  - elf no. 0moves, coordinates: 3 7
  - ...it moves until
  - elf no. 0moves, coordinates: 1 8
  - elf no.0 sent to the factory number0
  - elf no.1 coordinates: 1,2
  - factory no.0has elf[id: 0, id: 1]
  - elf no. 0moves, coordinates: 0 8
  - elf no. 1moves, coordinates: 1 1
  - elf no. 1waits at the barrier
  - elf no. 0moves, coordinates: 1 8
  - elf no. 0moves, coordinates: 0 8
  - elf no. 0moves, coordinates: 0 7
  - elf no. 0moves, coordinates: 1 7
  - elf no. 0moves, coordinates: 0 7
  - elf no. 0moves, coordinates: 0 6
  - elf no. 0moves, coordinates: 0 5
  - elf no. 0moves, coordinates: 0 4
  - elf no. 0moves, coordinates: 0 3
  - elf no. 0moves, coordinates: 1 3
  - elf no. 0moves, coordinates: 1 2
  - elf no. 0waits at the barrier
  - elf no.1 sent to the factory number0
  - elf no.2 coordinates: 6,4
  - factory no.0has elf[id: 0, id: 1, id: 2]
  - elf no. 2moves, coordinates: 6 5
  - elf no. 2waits at the barrier
  - elf no.2 sent to the factory number0
  - elf no.3 coordinates: 7,7
  - factory no.0has elf[id: 0, id: 1, id: 2, id: 3]
  - elf no. 3moves, coordinates: 7 8

- elf no. 3waits at the barrier
- elf no.3 sent to the factory number0
- elf no.4 coordinates: 4,5
- factory no.0has elf[id: 0, id: 1, id: 2, id: 3, id: 4]
- elf no. 4moves, coordinates: 4 4
- elf no. 4waits at the barrier
- clean up job after all tasks are done.
- elf no. 0moves, coordinates: 1 2
- elf no. 0waits at the barrier
- elf no. 2moves, coordinates: 6 6
- elf no. 4moves, coordinates: 5 4
- elf no. 1moves, coordinates: 0 1
- elf no. 1waits at the barrier
- elf no. 2waits at the barrier
- elf no. 3moves, coordinates: 7 9
- elf no. 4waits at the barrier
- elf no. 3moves, coordinates: 8 9
- elf no. 3waits at the barrier
- clean up job after all tasks are done.
- ...and so on

For the extra task 4

- I have kept only the following classes: Workshop, Factory, Elf, ElfSpawner.
- I added a class MyOwnCyclicBarrier: its constructor gets a number of parties. If the parties waiting variable is not zero, then the thread waits. Else, it means that all the parties reached the barrier, so “it gets lifted” - the number of parties waiting is reseted and all the threads waiting are notified to resume. The CyclicBarrierEvent enters.
- Another class added is CyclicBarrierEvent, which is basically a thread. Its run method contains only a printing message: “clean up job after all tasks are done.”
- All the other observations from Extra task 3 are still standing.

EXPLAINING THE OUTPUT FOR THE MAIN PROGRAM

As I am trying to check and to explain the output, I will create only 20 elves, each trying to move 1500 times, so I can get maximum 1500 gifts.

2 factories

Factory 0 has the dim 357

Factory 1 has the dim 498 the factories have the dimension requested

elf no.0 coordinates: 49,75

factory no.1has elf[id: 0]

elf no.1 coordinates: 191,354

factory no.0has elf[id: 1]

elf no.2 coordinates: 324,334

factory no.0has elf[id: 1, id: 2]

elf no.3 coordinates: 56,227

factory no.0has elf[id: 1, id: 2, id: 3] the elf registers to its assigned factory

elf no.4 coordinates: 329,44

factory no.0has elf[id: 1, id: 2, id: 3, id: 4]

elf no.5 coordinates: 38,466

factory no.1has elf[id: 0, id: 5]

elf no.6 coordinates: 331,145

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6]

elf no.7 coordinates: 107,192

factory no.1has elf[id: 0, id: 5, id: 7]

elf no.8 coordinates: 287,96

factory no.1has elf[id: 0, id: 5, id: 7, id: 8]

elf no.9 coordinates: 54,40

factory no.1has elf[id: 0, id: 5, id: 7, id: 8, id: 9]

elf no.10 coordinates: 271,42

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10]

elf no.11 coordinates: 65,146

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11]

elf no.12 coordinates: 288,245

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11, id: 12]

elf no.13 coordinates: 163,26

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11, id: 12, id: 13]

elf no.14 coordinates: 323,107

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11, id: 12, id: 13, id: 14]

elf no.15 coordinates: 314,281

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11, id: 12, id: 13, id: 14, id: 15]

elf no.16 coordinates: 114,261

factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11, id: 12, id: 13, id: 14, id: 15, id: 16]

elf no.17 coordinates: 305,89
factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11, id: 12, id: 13, id: 14, id: 15, id: 16, id: 17]
elf no.18 coordinates: 125,275
factory no.0has elf[id: 1, id: 2, id: 3, id: 4, id: 6, id: 10, id: 11, id: 12, id: 13, id: 14, id: 15, id: 16, id: 17, id: 18]
elf no.19 coordinates: 36,251
factory no.1has elf[id: 0, id: 5, id: 7, id: 8, id: 9, id: 19]
the elf no. 0 created 1500
the elf no. 1 created 1500
the elf no. 2 created 1500
the elf no. 3 created 1500
the elf no. 4 created 1500
the elf no. 5 created 1500
the elf no. 6 created 1500
the elf no. 7 created 1500
the elf no. 8 created 1500
the elf no. 9 created 1500
the elf no. 10 created 1500
the elf no. 11 created 1500
the elf no. 12 created 1500
the elf no. 13 created 1500
the elf no. 14 created 1495 the elf was not able to move 5 times because of other elves
the elf no. 15 created 1500
the elf no. 16 created 1500
the elf no. 17 created 1499
the elf no. 18 created 1500
the elf no. 19 created 1500

Adding the **elf retirer**, which will only retire 5 elves, the output would have, beside what I have specified already, the following:

```
elf no.0 has retired
the elf no.0 created 501gifts before it was retired.                    It got to move 501 times
factory no.0has retired the elf no.0                                    which elf will be removed from the list
now it has:[ id: 1,  id: 3,  id: 12,  id: 13,  id: 15,  id: 16,  id: 19]    deletes the elf from the list
elf no.1 has retired
the elf no.1 created 641gifts before it was retired.
factory no.0has retired the elf no.1
now it has:[ id: 3,  id: 12,  id: 13,  id: 15,  id: 16,  id: 19]
elf no.18 has retired
the elf no.18 created 374gifts before it was retired.
factory no.1has retired the elf no.18
now it has:[ id: 5,  id: 7,  id: 8,  id: 9,  id: 10]
elf no.10 has retired
the elf no.10 created 723gifts before it was retired.
factory no.1has retired the elf no.10
now it has:[ id: 5,  id: 7,  id: 8,  id: 9]
elf no.7 has retired
the elf no.7 created 954gifts before it was retired.
factory no.1has retired the elf no.7
now it has:[ id: 5,  id: 8,  id: 9]
```

Now will enter the producers - elves and the consumers - reindeers. Even if the output is not very helpful, it shows that all the reindeers enter in the factory and get gifts to send to santa. As I did not use a list with a fixed capacity, I cannot check properly if this is 100% correct.

```
the reindeer no. 0 has received: 35318
the reindeer no. 5 has received: 35533
the reindeer no. 3 has received: 33601
the reindeer no. 1 has received: 35512
the reindeer no. 8 has received: 35174
the reindeer no. 4 has received: 31780
the reindeer no. 6 has received: 31884
the reindeer no. 7 has received: 31574
the reindeer no. 2 has received: 40421
the reindeer no. 0 has received: 155407
the reindeer no. 2 has received: 162258
the reindeer no. 7 has received: 139190
the reindeer no. 6 has received: 135328
the reindeer no. 5 has received: 144424
the reindeer no. 4 has received: 136037
```

the reindeer no. 3 has received: 144330
the reindeer no. 1 has received: 144602
the reindeer no. 8 has received: 149286
the reindeer no. 0 has received: 293424
the reindeer no. 7 has received: 260840

Entering the producers - reindeers and the consumer - Santa. Now I am using a blocking queue with a fixed capacity, maximum 1000 gifts. The snippet of the output is:

the reindeer no. 6 has received: 0
Santa has received: 1000 Santa has already 1000 gifts because probably the other reindeers did not get to say what they received yet
the reindeer no. 8 has received: 884
Santa has received: 2000
the reindeer no. 7 has received: 1297
the reindeer no. 6 has received: 835
the elf no. 0 created 515 gifts.
Santa has received: 3000
the reindeer no. 4 has received: 2598
the reindeer no. 0 has received: 2357
the reindeer no. 2 has received: 4510
the reindeer no. 1 has received: 4530
Santa has received: 4000
the reindeer no. 5 has received: 3967
the reindeer no. 3 has received: 3682
Santa has received: 5000
Santa has received: 6000

MY OBSERVATIONS

It was a challenging task, as I am not an expert in the field, but I hope I got to help Santa at least a little.

As I have tried to check the outputs on small numbers, I did not notice problems.

The *first big* task is segmented in a lot of smaller ones, some of them were not reached, as:

- Every few seconds the factory will ask all its elves their position in the factory: as I am already sending the locations of the elves when one of them sends a gift, I did not implement this function. I understood that the Elf class should have been also synchronized for this, and I did not get how and why.
- the factory will ask all elves to pass on the information on their location to give it to Santa: as I am understanding this task, I already implemented the part of "will ask all elves to pass their location", but I did not send it to Santa, as his job (in my program) is only to read the gifts the reindeers send to him…
- A random time must pass between two consecutive factory readings: in my program, when they get the chance to enter the factory and there are gifts, they go in.

For all the others, I have tried to think them as good as I could. The small checks I have done seemed ok to me.

I hope Santa is happy with my help. Though, I hope that maybe next time he requests it I'll do better, as I hope to have more experience regarding threads in Java.