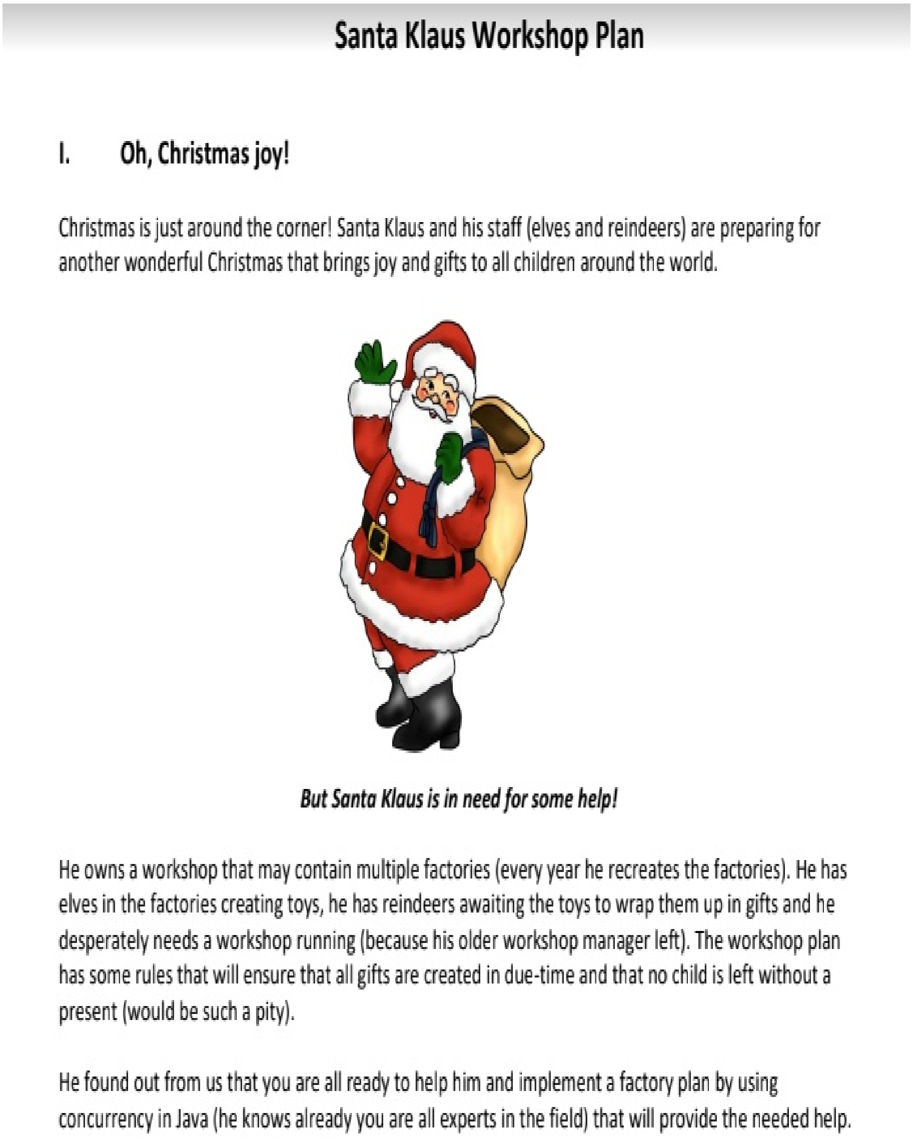
Concurrent and Distributed Systems Homework

Santa Klaus Workshop Plan

Buzdugă Ionuţ Gabriel
Group CEN 3.1B
Year 3
Computer and Information Technology-English

1 Problem Statement



Santa Klaus Workshop Plan

I. Oh, Christmas joy!

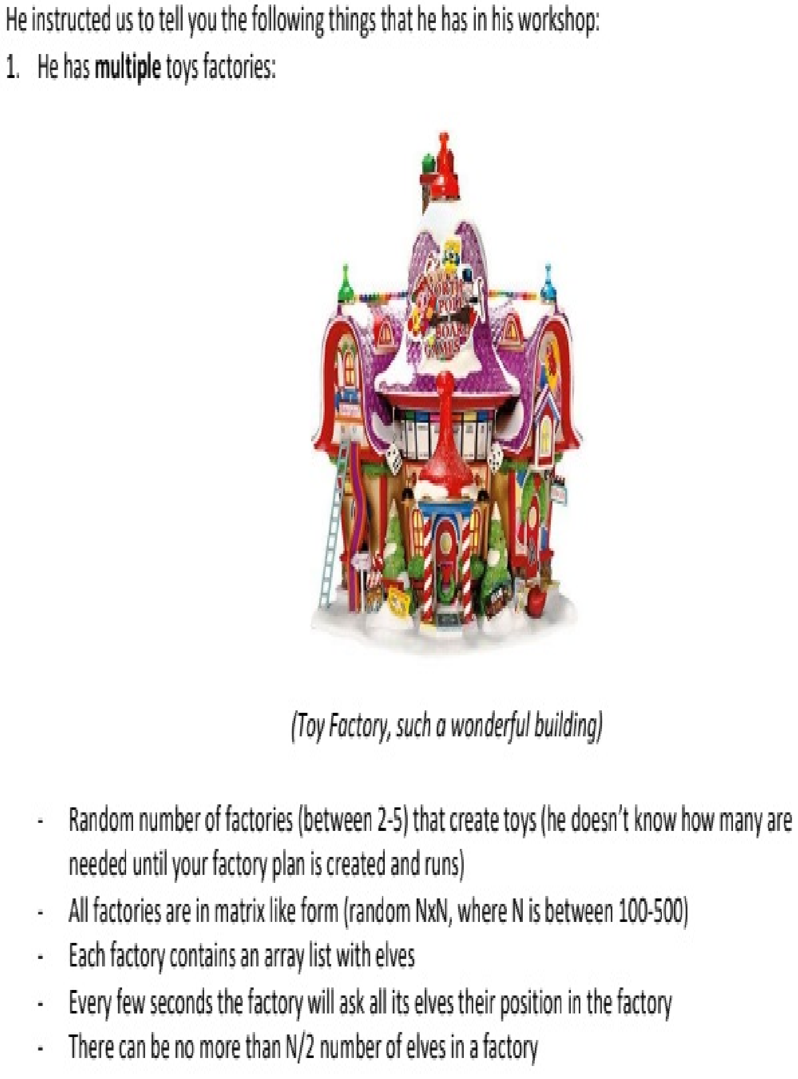
Christmas is just around the corner! Santa Klaus and his staff (elves and reindeers) are preparing for another wonderful Christmas that brings joy and gifts to all children around the world.

But Santa Klaus is in need for some help!

He owns a workshop that may contain multiple factories (every year he recreates the factories). He has elves in the factories creating toys, he has reindeers awaiting the toys to wrap them up in gifts and he desperately needs a workshop running (because his older workshop manager left). The workshop plan has some rules that will ensure that all gifts are created in due-time and that no child is left without a present (would be such a pity).

He found out from us that you are all ready to help him and implement a factory plan by using concurrency in Java (he knows already you are all experts in the field) that will provide the needed help.

2 Problem Requirements



He instructed us to tell you the following things that he has in his workshop:

1. He has **multiple** toys factories:

(Toy Factory, such a wonderful building)

- Random number of factories (between 2-5) that create toys (he doesn't know how many are needed until your factory plan is created and runs)
- All factories are in matrix like form (random NxN, where N is between 100-500)
- Each factory contains an array list with elves
- Every few seconds the factory will ask all its elves their position in the factory
- There can be no more than N/2 number of elves in a factory

2. He has **elves**. Elves are really important; they create all the nice toys. Here's what he told us about them:

(Happy elf working hard)

- Elves are spawning randomly in each factory at a random place in the factory (random time between 500-1000 milliseconds)
- When elves are spawned randomly they need to tell the factory that they are there
- No two elves can be on the same position
- Each elf works independently of any other elf
- Elves create gifts by:

  - Moving in one direction (left, right, up, down)
  - When they reach a wall of the factory they move in any other direction than the wall's
  - Once they move they also create the gift
  - If an elf is surrounded by other elves and cannot move it stops for a random time (between 10-50 milliseconds)
  - When the gift is created they update their factory to know what gift they created and the factory will ask all elves to pass on the information on their location to give it to Santa
  - After the elf creates a gift, he needs to rest for a short while (30 milliseconds)
    *) They are really fast
  - Elves will work like in... forever (well, at least until December 25th. As such they don't need to stop.

3. He also has **reindeers**:

(Awaiting reindeer, wondering if he receives all gifts to pass to Santa)

- Reindeers await to receive gifts from the factory. Reindeers are available in the workshop
- They can always read from the factories the number of gifts (and which gifts), but they won't be allowed access when elves notify the factories about new gifts or when the factory itself asks the elves about new gifts
- A maximum of 10 reindeers can read from a factory at a time
- A random time must pass between two consecutive factory readings
- Reindeers will put all gifts through a pipe to give to Santa
- Multiple reindeers at the same time put the gifts while Santa reads by itself all gifts
- Reindeers are known from the beginning (there are more than 8, because Santa needs backup).

4. The main Santa's Workshop will do the following:

(Sign to Santa's workshop hidden at North Pole)

- this workshop contains all factories
- it creates the factories
- it also spawns elves and gives them a random factory to work in (remember that each elf is responsible to register himself to the factory)

3 Implementation Process

In this section it will be presented the thoughts, solutions, and choices that I made for the Implementation of the problem.

The programs runs successfully using 2 java applications.

3.1 The first one manages all the workshop functionalities and contains the following classes:

- Elf
- ElfGenerator
- ElfLocation
- ElfRetire
- Factory
- Gift
- GiftQueue
- IGiftQueue
- OwnCyclicBarrier
- Reindeer
- Workshop

3.2 The second one manages the TCP/IP server connection between Santa and his Workshop and contains the classes:

- Connection
- WorkshopServer

Elf Class: Class used for the creation of elf Threads which have a runnable method that manages the elf movement, creation of gifts , barrier requirements for the elves and elf Retirement.

When a new Elf thread is created, a check up is made where its position is regulated.If there are less then N/2 elves in the factory, the elf is placed on a new position and then it starts working. The moveElf() method is the most important one that manages how the elf moves in relation to the factory bounds and the other elves and also when the elf creates a presents and rests through the following methods: createGift(), createPresentRest(), sleepWhenNotMoving(), checkIfOnDiagonalAndRestOrJump(), tryToRetireElf()

ElfGenerator Class:

It has one runnable method that waits a certain amount of time before creating an elf and then places that elf in a random factory array from the ones that were created.

ElfLocation Class:

Contains the x(for rows) and y(for columns) coordinates which will be used in order to find an elf inside a factory

ElfRetire Class:

This is The class that implements elf Retirement requirement(Task I). It has a runnable method that releases a semaphore after a certain amount of time.Each of the elf thread instances then try to aquire this semaphore and if they succeed they will be retired.

Factory Class:

This is the class that manages elf generation and position in the factory. The addElfToFactory() method checks to see if there is enough space in the factory for the elf to be generated and if there is, the method will start generating random valid positions from the factory and if there aren't any other elves there, the new generated elf will be placed in a HashMap (elf map position) to further manage it placement for the future movements

Gift class:

The gift class manages the position of gifts, which will be created by the elves and received by the reindeers

GiftQueue class:

Because the gifts need to be received in a certain way by the reindeers and then sent to Santa one at a time, we needed a queue where all the gifts will be managed with the use of pushGift() and popGift() methods

IGiftQueue class:

This is just an interface of the GiftQueue class which groups its abstract methods with empty bodies.

OwnCyclicBarrier class:

This is the barrier class created for Task 4, with the same specs as in the Task requirement. We have a constructor with the parties parameter which counts how many of the elves reached the barrier state. The implemented await() method is used so that once an elf enters the waiting state, he need to wait for all the other elves needed to reach that state. When all the waiting states reach the number of the parties the barrier resets using the doWait() method

Reindeer class:

This is a class which manages the Reindeers. The reindeers which have a maximum working number make the connection to Santa using the TCP/IP server connection. Using the input and output flux created and the access to the GiftQueue They take the gifts that are inside the queue individually and transport them using the server connection to Santa, which keeps track of the total number of presents received from its reindeers.

Workshop class:

This is the first main class that manages one of the java applications Implemented.

Here we have stated all the restrictions imposed by the problem requirements: Number of factories(2-5),Factory Size:100-500, Number of Reindeers:8-12

The main method is implemented here:

Using a random variable rand we get the number of our factories and reindeers. For each factory that was created a new GiftQueue instance is also created. The run methods for the elf generation and elf retirement are also started. And also the factories, elf generation, elf retirement and reindeers are Joined.

The Connection Class and second java Application Class:WorkshopServer

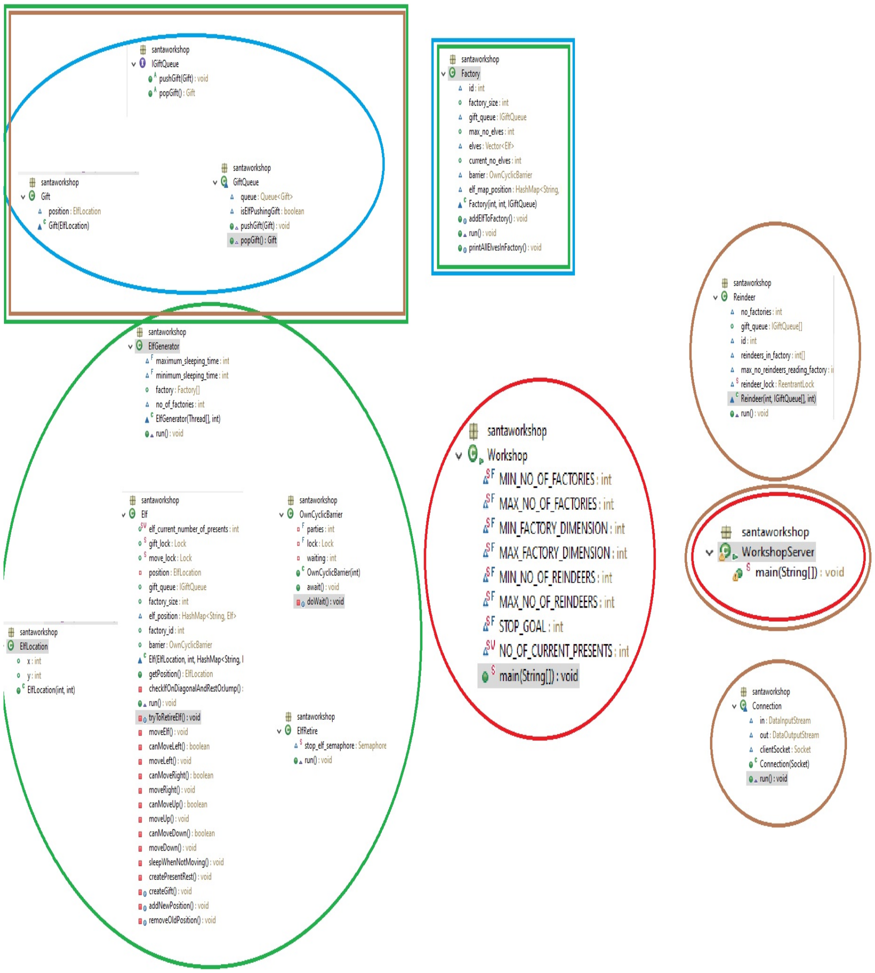
This are the classes implemented for the TCP/IP server connection. These classes were explained and implemented in Laboratory 12 of the Concurrent and Distributed Systems Course. The most important one Is the **WorkshopServer** which is a main java application class that need to be ran before running the other main application in this project(Workshop).

From how I designed the implementation the WorkshopServer needs to be running while we start the Workshop class in order to make the connection successfully.And after it connects a message will appear when trying to run the WorkshopServer apllication again:Listen socket:Address already in use: bind, meaning that the server managed the connection.

4 Application Outline

4.1 High Level Application Structure

In this section we review the arhitectural structure of our application.



🟥 Main apllications that need to be executed in this order:
1)WorkshopServer
2)Workshop(while the 1st one is still running)

🟩 Classes that manage elf actions

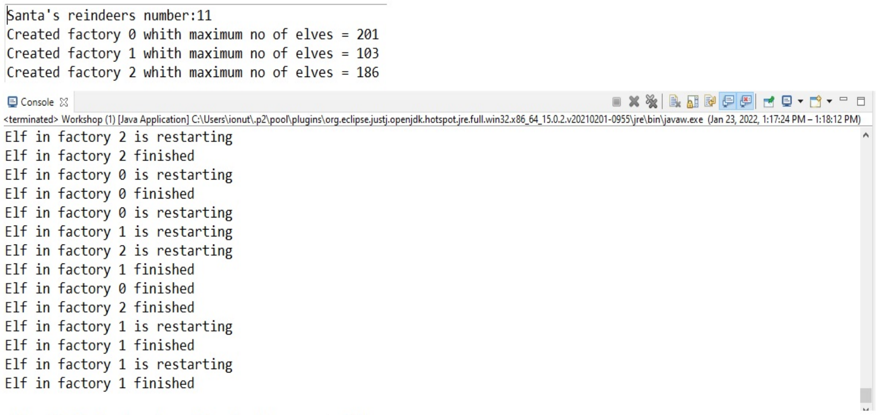
🟦 Classes that manage Gifts

🟫 Classes that manage reindeer actions

4.2 Data Testing

For testing the Data I followed the aproach of setting a Maximum amount of presents that needed to be reached(instead of the factory running forever), and seeing how fast the workshop reaches that goal when having either more factories or more reindeers.

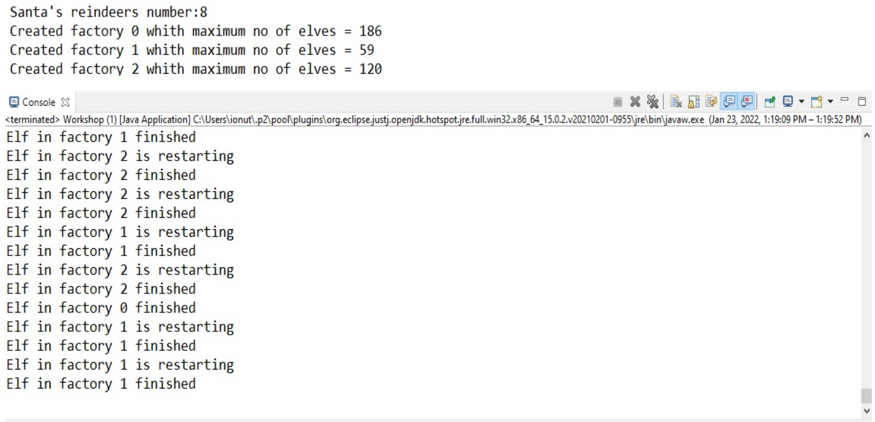
Test1:3 factories 11 reindeers 48 s



```
Santa's reindeers number:11
Created factory 0 whith maximum no of elves = 201
Created factory 1 whith maximum no of elves = 103
Created factory 2 whith maximum no of elves = 186
```

```
<terminated> Workshop (1) [Java Application] C:\Users\ionut\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Jan 23, 2022, 1:17:24 PM – 1:18:12 PM)
Elf in factory 2 is restarting
Elf in factory 2 finished
Elf in factory 0 is restarting
Elf in factory 0 finished
Elf in factory 0 is restarting
Elf in factory 1 is restarting
Elf in factory 2 is restarting
Elf in factory 1 finished
Elf in factory 0 finished
Elf in factory 2 finished
Elf in factory 1 is restarting
Elf in factory 1 finished
Elf in factory 1 is restarting
Elf in factory 1 finished
```

Test2:3 factories, 8 reindeers =>43 s



```
Santa's reindeers number:8
Created factory 0 whith maximum no of elves = 186
Created factory 1 whith maximum no of elves = 59
Created factory 2 whith maximum no of elves = 120
```

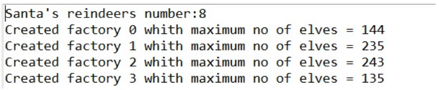
```
<terminated> Workshop (1) [Java Application] C:\Users\ionut\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Jan 23, 2022, 1:19:09 PM – 1:19:52 PM)
Elf in factory 1 finished
Elf in factory 2 is restarting
Elf in factory 2 finished
Elf in factory 2 is restarting
Elf in factory 2 finished
Elf in factory 1 is restarting
Elf in factory 1 finished
Elf in factory 2 is restarting
Elf in factory 2 finished
Elf in factory 0 finished
Elf in factory 1 is restarting
Elf in factory 1 finished
Elf in factory 1 is restarting
Elf in factory 1 finished
```

On the first two tests I compared two situations:1st when we have 3 factories and 11 reindeers, Second when we have 3 factories and 8 reindeers.

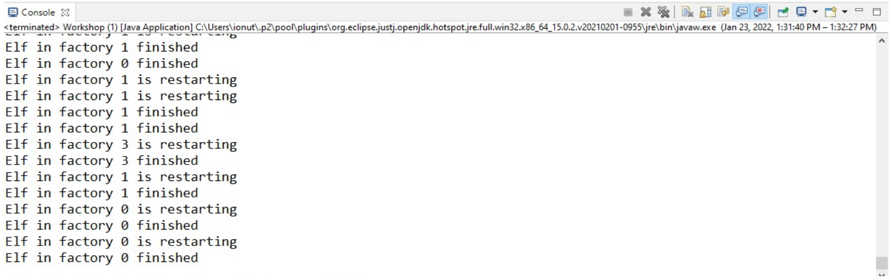
The main observation for this test is that when we have less reindeers the application reached the 1000 gifts goal faster (42s) rather then (48s) when we had 11 reindeers.

this is probably because the Queue is less crowded when we dont have the maximum amount of reindeers, so then they dont need to wait for the queue to have a free spot.The results are also influenced by the maximum number of elves in each factory, in the second case having less elves per total.

Test3:4 factories, 8 reindeers, 47 s

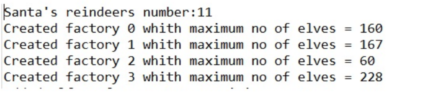


```
Santa's reindeers number:8
Created factory 0 whith maximum no of elves = 144
Created factory 1 whith maximum no of elves = 235
Created factory 2 whith maximum no of elves = 243
Created factory 3 whith maximum no of elves = 135
```

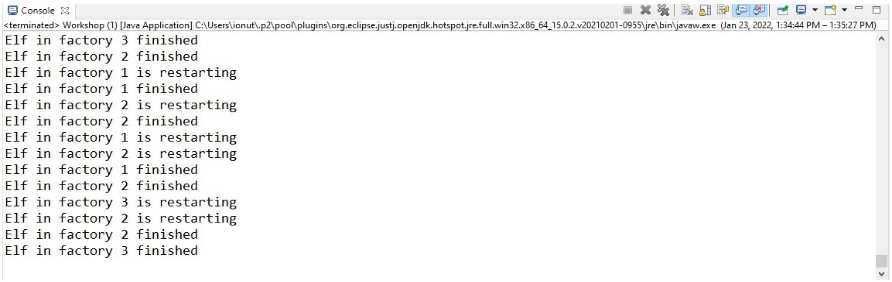


```
<terminated> Workshop (1) [Java Application] C:\Users\ionut\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  (Jan 23, 2022, 1:31:40 PM – 1:32:27 PM)
Elf in factory 1 finished
Elf in factory 0 finished
Elf in factory 1 is restarting
Elf in factory 1 is restarting
Elf in factory 1 finished
Elf in factory 1 finished
Elf in factory 3 is restarting
Elf in factory 3 finished
Elf in factory 1 is restarting
Elf in factory 1 finished
Elf in factory 0 is restarting
Elf in factory 0 finished
Elf in factory 0 is restarting
Elf in factory 0 finished
```

Test4:4factories, 11 reindeers, 43 s



```
Santa's reindeers number:11
Created factory 0 whith maximum no of elves = 160
Created factory 1 whith maximum no of elves = 167
Created factory 2 whith maximum no of elves = 60
Created factory 3 whith maximum no of elves = 228
```



```
<terminated> Workshop (1) [Java Application] C:\Users\ionut\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  (Jan 23, 2022, 1:34:44 PM – 1:35:27 PM)
Elf in factory 3 finished
Elf in factory 2 finished
Elf in factory 1 is restarting
Elf in factory 1 finished
Elf in factory 2 is restarting
Elf in factory 2 finished
Elf in factory 1 is restarting
Elf in factory 2 is restarting
Elf in factory 1 finished
Elf in factory 2 finished
Elf in factory 3 is restarting
Elf in factory 2 is restarting
Elf in factory 2 finished
Elf in factory 3 finished
```

The same behaviour we observe in the next set of tests where now we have 4 factories.

The case where we have less elves is faster than the one with more elves.

This is because the sleeping time of elves is adding up the more elves there are in the factories

The overall Speed increases from the previous tests because of the additional factory but not by that much.

Conclusions

Working on this problem helped me understand more about the course and the importance of concurrency problems.

There are multiple new things that I learned while working on this project:

1. First of all,it helped me to improve my coding ability by interacting with new types of ways to solve problems.

2. Secondly,I believe another key factor that has improved my ability to solve problems is having to make a report about the problem I am working with.

   Being able to enunciate a problem in a formal way can undoubtedly improve our knowledge about the problem,but also about programming in general.

   The main factors of this are in my opinion The application outline and the experimental data sections.

   The first one has a big role in helping how to structure the algorithm in form of interactions of classes,methods and other newly introduced concepts learned at this year's course such as concurrent programming using Threads, barrier Cycles, server connections, semaphores ,monitors and locks.

Meanwhile,the latter helps to verify that our problem works on multiple sets of data and the output data is clearly developed to both me and the reader.

4.3 References

1. GeeksforGeeks.com
2. StackOverflow.com
3. The chapters and laboratories of the Concurrent and Distributed Systems course .