Universitatea din Craiova
Facultatea de Automatică, Calculatoare și Electronică

Atelierul lui Mos Craciun

Sisteme concurente si distribuite

Studentă: Bucă-Ghebaură Elizabetha Alexandrina

Specializarea: Calculatoare Română
Grupa: CR 3.1A
An: III

Ianuarie 2022

Introducere cuprins

Abstract

Acest raport este o introducere in obiectivele de dezvoltare a temei de casa la disciplina Inteligenta Artificiala. Documentul contine, de asemenea, o descriere a livrabilelor.

1 Enuntul problemei - pe scurt

Craciunul se apropie! Mos Craciun si angajatii sai (elfii si renii) se pregatesc pentru un nou Craciun care aduce bucurie si cadouri tuturor copiilor din jurul lumii.

Dar Mos Craciun are nevoie de ajutor!

Mos Craciun stie ca il puteti ajuta. El stie ca aveti la dispozitie urmatoarele instrumente utile de programare concurenta:

- Semafoare, monitoare si zavoare

- Fire de executie. Fiecare elf poate fi reprezentat de un fir separate de executie in cadrul planului

- Ele trebuie sa comunice: elfii creaza cadouri si renii le primesc pentru a le transmite lui Mos Craciun care le va livra copiilor

- Mos Craciun vrea ca renii sa-I transmita cadourile prin TCP/IP, pentru a nu se pierde nici un cadou.

Sarcini suplimentare:

- Retragerea unui elf

- "Odihnirea" unui elf

- Folosirea clasei CyclicBarrier

- Implementarea proprie a barierei

2 Descrierea si intelegerea problemei

Stim ca Mos Craciun detine mai multe fabrici de jucarii, iar in fiecare fabrica exista o multime de elfi. Aceasta fabrica creeaza jucarii, care sunt create de catre elfi.

Elfii sunt creati aleator in fiecare fabrica intr-o pozitie aleatoare; imediat dupa ce a fost creat, fiecare dintre ei trebuie sa isi raporteze pozitia unde se afla (2 elfi nu pot fi in aceeasi pozitie), iar fiecare lucreaza independent.

Elfii creeaza cadouri mutandu-se in directiile sus, jos, stanga, dreapta, iar la fiecare mutare ei creeaza un cadou. Dupa ce s-a creat cadoul, fiecare va informa fabrica asupra a ce cadou s-a creat, iar apoi fabrica va interoga toti elfii pentru a comunica locatia lor, urmand apoi ca aceasta sa fie transmisa lui Mos Craciun.

Stim si ca Mos Craciun are multi reni. Acestia asteapta sa primeasca cadourile de la fabrici, iar ei sunt disponibili in atelier. Renii vor transmite toate cadourile spre Mos Craciun.

Atelierul contine toate fabricile, le creeaza si creeaza de asemenea si elfii, aleator in fabrici, iar fiecare elf se va inscrie la fabrica in care a fost creat.

Am reprezentat fiecare elf printr-un fir separat de executie in cadrul planului atelierului.

Aceste fire comunica, astfel:

⋆ elfii creeaza cadouri

⋆ renii primesc cadourile, apoi le transmit lui Mos Craciun care a le livra copiilor

Asadar vom avea 9 clase pentru inceput (inainte de a implementa si clasa Cyclic Barrier din sectiunea Sarcini suplimentare), astfel:

⋆ un main cu ajutorul caruia programul nostru va functiona

⋆ atelierul si fabrica de jucarii, precum si transferarea cadourilor

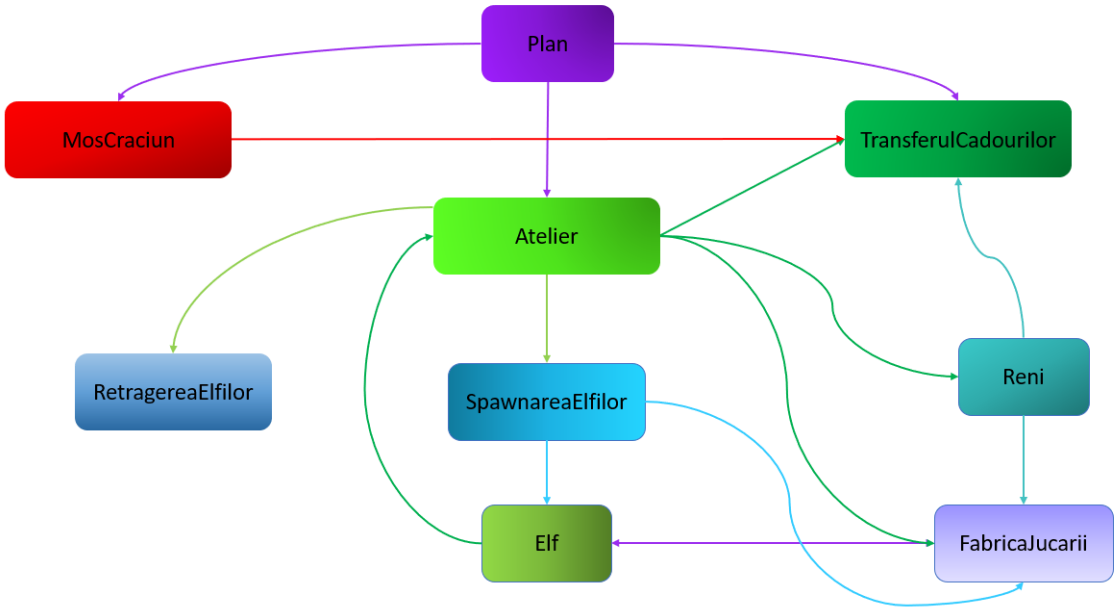
⋆ vom avea clasele Mos Craciun, Elf, Reni

⋆ clasele pentru retragerea si spawnarea elfilor

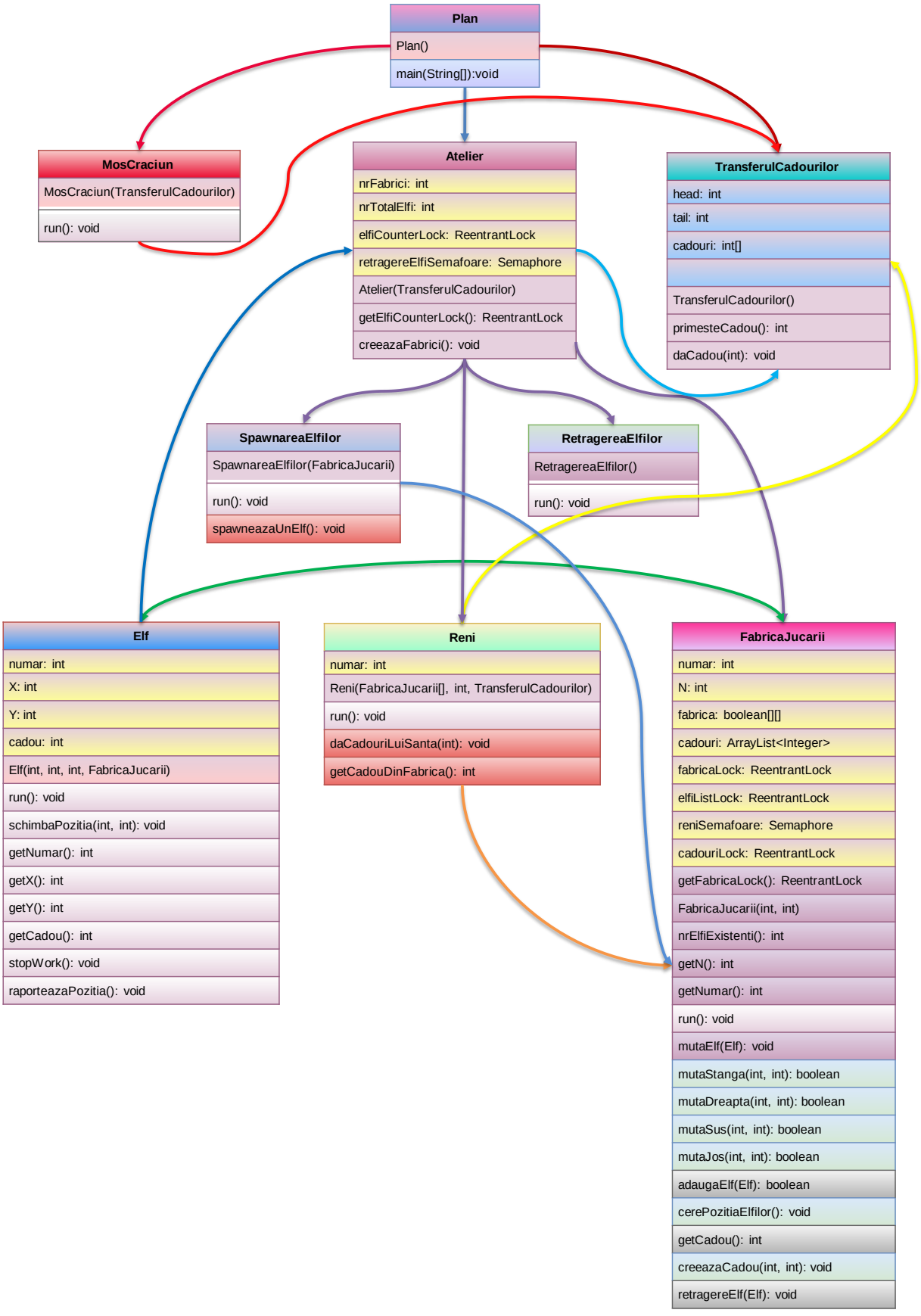
3 Proiectarea aplicației experimentale

3.1 Structura de nivel inalt a aplicatiei

Aceasta figura reprezinta programul meu cu toate clasele create.
In ***Plan.java*** sunt apelate functiile problemei, iar in restul claselor sunt implementate toate functiile problemei.



De asemenea, diagrama UML de mai jos reprezinta o figura mai ampla a programului, unde se pot observa clasele implementate, alaturi de functii si parametrii.


Plan
Plan()
main(String[]):void
MosCraciun
MosCraciun(TransferulCadourilor)
run(): void
Atelier
nrFabrici: int
nrTotalElfi: int
elfiCounterLock: ReentrantLock
retragereElfiSemafoare: Semaphore
Atelier(TransferulCadourilor)
getElfiCounterLock(): ReentrantLock
creeazaFabrici(): void
TransferulCadourilor
head: int
tail: int
cadouri: int[]
TransferulCadourilor()
primesteCadou(): int
daCadou(int): void
SpawnareaElfilor
SpawnareaElfilor(FabricaJucarii)
run(): void
spawneazaUnElf(): void
RetragereaElfilor
RetragereaElfilor()
run(): void
Elf
numar: int
X: int
Y: int
cadou: int
Elf(int, int, int, FabricaJucarii)
run(): void
schimbaPozitia(int, int): void
getNumar(): int
getX(): int
getY(): int
getCadou(): int
stopWork(): void
raporteazaPozitia(): void
Reni
numar: int
Reni(FabricaJucarii[], int, TransferulCadourilor)
run(): void
daCadouriLuiSanta(int): void
getCadouDinFabrica(): int
FabricaJucarii
numar: int
N: int
fabrica: boolean[][]
cadouri: ArrayList<Integer>
fabricaLock: ReentrantLock
elfiListLock: ReentrantLock
reniSemafoare: Semaphore
cadouriLock: ReentrantLock
getFabricaLock(): ReentrantLock
FabricaJucarii(int, int)
nrElfiExistenti(): int
getN(): int
getNumar(): int
run(): void
mutaElf(Elf): void
mutaStanga(int, int): boolean
mutaDreapta(int, int): boolean
mutaSus(int, int): boolean
mutaJos(int, int): boolean
adaugaElf(Elf): boolean
cerePozitiaElfilor(): void
getCadou(): int
creeazaCadou(int, int): void
retragereElf(Elf): void

3.2 Specificarea datelor de intrare

Ca si date de intrare avem:
* **numarul si dimensiunea fabricilor**
* **renii**
* **elfii**

Toate aceste date de intrare nu sunt date de la tastatura, ci sunt generate random cu ajutorul bibliotecii importate, **java.util.Random**.

Voi da mai jos cateva exemple de secvente de cod folosite, utilizand biblioteca Random:

*func*tion **getCadouDinFabrica()**

```
1.      Random rand = new Random();
2.      int fabrica = rand.nextInt(Atelier.nrFabrici) + 0;
3.      return fabrici[fabrica].getCadou();
end function
```

*func*tion **stopWork()**

```
1.      Random rand = new Random();
2.      long milis = rand.nextInt(50) + 10;
end function
```

*func*tion **creeazaFabrici()**

```
1.      Random rand = new Random();
2.      nrFabrici = rand.nextInt(4) + 2;
3.      int nrReni = rand.nextInt(10) + 8;
end function
```

3.3 Specificarea ieșirilor/rezultatelor

Dupa cum se vede in figurile cu output-ul, incepe transferarea cadourilor.

⋆ Pentru output-ul **Cyclic Barrier**:

Mai intai se creeaza fabricile si renii, apoi aflam ce dimensiune are fiecare fabrica din cele create. Astfel, avem si elfii care se creeaza la pozitii aleatorii in fiecare fabrica, apoi fiecare dintre ei creeaza cate un cadou.

Urmeaza ca un ren aleatoriu sa primeasca cadoul facut de elful respectiv si sa i-l dea lui Mos Craciun. Mos Craciun pune acest cadou in sac, iar apoi trebuie ca elfii sa se retraga de la fabricile aferente fiecaruia, urmand sa se creeze altii in locul lor (in pozitii diferite fata de cei dinainte) si se continua la fel la infinit.

⋆ Pentru output-ul **Own Barrier**:

De asemenea, si aici mergem pe acelasi principiu: vedem dimensiunea si numarul fiecarei fabrici, numarul renilor, pozitiile fiecarui elf si fabrica unde este creat. Elfii creeaza cadourile, renii le primesc si apoi le dau lui Mos Craciun care le pune in sac. Singura diferenta este ca elfii nu se mai retrag dupa ce au dat cadoul renilor, ci isi schimba pozitia si creeaza din nou cadouri pe care le ofera renilor, apoi iar isi schimba pozitia si asa mai departe.

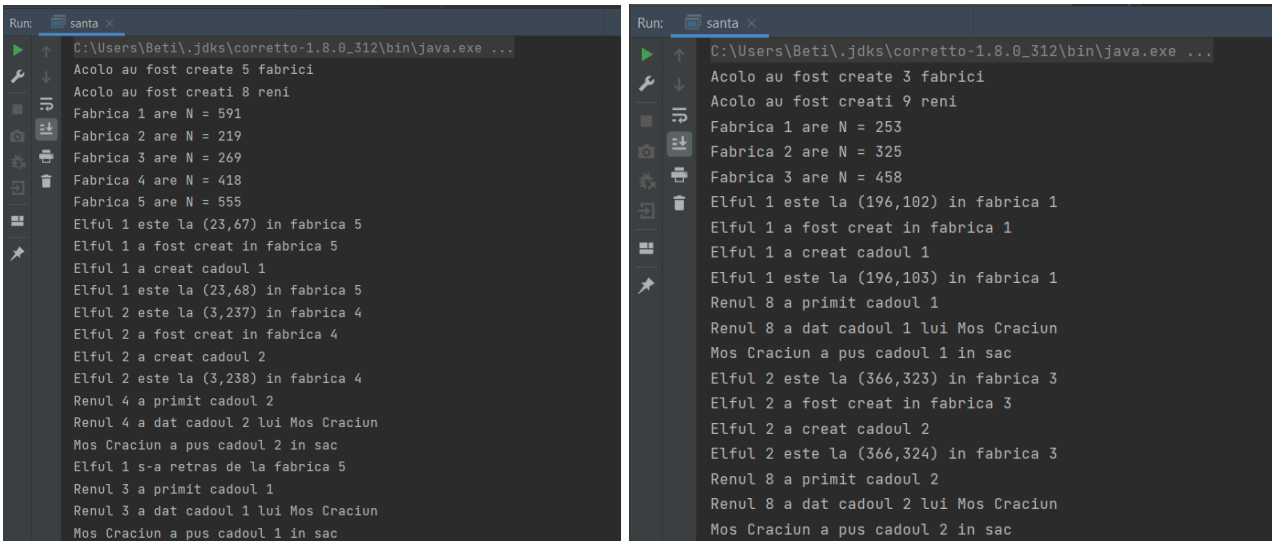
⋆ Pentru output-ul **Semafoare**:

Si aici se intampla la fel ca si in celelalte 2 cazuri. (se creeaza fabricile, renii, elfii etc).

Se intampla la fel ca si la OwnBarrier, adica elfii nu se mai retrag de la fabrici, ci dupa ce fiecare este creat si isi are pozitia in fabrica, el va dormi pe aceeasi pozitie si apoi va crea cadoul pe care i-l va da mai tarziu renului.

Nota:

Figurile despre care s-a vorbit mai sus se afla pe pagina urmatoare.



(a) (b)

Figure 1: Ouput Cyclic Barrier / Own Barrier

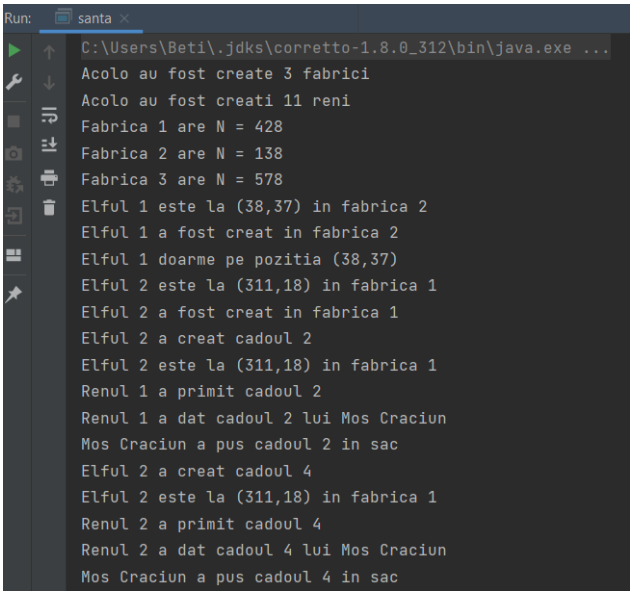


Figure 2: Ouput Semaphores

3.4 Lista tuturor modulelor aplicatiei si descrierea lor

Aici voi realiza o mica prezentare a tuturor modulelor aplicatiei si o descriere a acestora, cu ce scop au fost folosite, de ce si la ce ne ajuta. Am precizat la subparagraful 3.1 care este structura de nivel inalt a aplicatiei, urmand acum sa specific lista cu toate modulele cat si descierea acestora.

Am ales ca pentru aceasta problema sa folosesc 9 surse/module/clase si anume cele enumerate mai jos:

- **Elf** - implementează un fir care se comportă ca un elf
- **RetragereaElfilor** - implementeaza un thread (fir) pe care il utilizam pentru a retrage un elf aleatoriu
- **SpawnareaElfilor** - implementeaza un thread (fir) folosit pentru a genera un elf intr-o anumita fabrica
- **TransferulCadourilor** - reprezinta o coada concurenta folosita ca mijloc de transfer intre Mos Craciun si reni
- **Plan** - punctul de plecare al programului
- **Reni** - implementeaza un thread care se comporta ca niste reni
- **MosCraciun** - implementeaza un thread care se va comporta ca Mos Craciun
- **FabricaJucarii** - implementeaza un thread care se va comporta ca o fabrica
- **Atelier** - Atelierul lui Mos Craciun

3.5 Lista tuturor funcțiilor aplicației, grupate pe module

3.5.1 Descrierea scopului functiei

- **Elf**
  - *run* - elful va executa urmatoarele actiuni intr-o bucla infinita:
    - creeaza un cadou nou
    - se muta in fabrica
    - doarme 30 milisecunde
    - incearca sa se retraga de la fabrica
  - *schimbaPozitia* - schimba coordonatele curente ale fiecarui elf
  - *getNumar* - returneaza numarul de identificare al elfului

- *getX* - returneaza coordonata X curenta a elfului
- *getY* - returneaza coordonata Y curenta a elfului
- *getCadou* - returneaza cadoul curent al elfului
- *stopWork* - il obliga pe elf sa doarma intre 10 si 50 milisecunde
- *raporteazaPozitia* - afiseaza pozitia curenta a elfului in matricea fabrica

- **RetragereaElfilor**
  - *run* - thread-ul va executa urmatoarele actiuni intr-o bucla infinita:
    - elibereaza un permis al semaforului de retragere pentru ca un elf sa se poata retrage dupa ce si-a terminat treaba
    - doarme 50 milisecunde

- **SpawnareaElfilor**
  - *run* - thread-ul va executa urmatoarele actiuni intr-o bucla infinita:
    - doarme intre 500 si 1000 milisecunde
    - spawneaza un elf
  - *spawneazaUnElf* - thread-ul va executa urmatoarele actiuni:
    - primeste lock-ul (zavorul) matricei din fabrica si o blocheaza
    - creeaza un nou elf daca este posibil (numarul de elfi existenti in fabrica trebuie sa fie mai mic decat dimensiunea fabricii / 2)
    - primește zavorul contorului elfilor din fabrica si il blocheaza astfel incat numarul total al elfilor sa nu poata fi modificat
    - adauga elful la fabrica si creste numarul total de elfi
    - deblocheaza zavorul pentru elfii fabricii
    - deblocheaza zavorul matricei din fabrica

- **TransferulCadourilor**
  - *primesteCadou* - metoda utilizata de Mos Craciun pentru a lua cadourile de la reni
  - *daCadou* - metoda utilizata de reni pentru a da cadourile lui Mos Craciun

- **Plan**
  - *main*:
    - creeaza coada de transfer de cadouri
    - il creeaza pe Mos Craciun
    - creeaza atelierul

        * atelierul incepe sa construiasca fabrici
        * Mos Craciun incepe sa primeasca cadourile de la reni

- **Reni**
  - *cadouQueue* - mijlocul de transferare a cadourilor
  - *run* - thread-ul va executa urmatoarele actiuni intr-o bucla infinita:
    * primeste un cadou de la fiecare fabrica
    * da cadourile lui Mos Craciun prin *cadouQueue*
    * doarme intre 10 si 30 milisecunde
  - *daCadouLuiSanta* - da un cadou lui Mos Craciun (care va fi pus in *cadouQueue*)
  - *getCadouDinFabrica* - intra intr-o fabrica random din cele existente deja si ia un cadou de acolo

- **MosCraciun**
  - *cadouQueue* - mijlocul de transfer al cadourilor
  - *run* - Mos Craciun va primi cadouri la infinit, fara oprire

- **FabricaJucarii**
  - *getFabricaLock* - returneaza zavorul matricei fabrica
  - *getN* - returneaza dimensiunea matricei din fabrica
  - *getNumar* - returneaza numarul fabricii
  - *run* - thread-ul va executa urmatoarele actiuni:
    * cere pozitia tuturor elfilor
    * doarme pentru 3000 milisecunde
  - *mutaElf* - muta un elf in fabrica
    * blocheaza zavorul matricei din fabrica
    * incearca sa se miste in orice directie sau nu mai merge daca este inconjurat
    * deplasarea intr-o directie inseamna schimbarea pozitiei in matrice, crearea unui cadou, modificarea pozitiei actuale a elfilor; de asemenea, mai inseamna si faptul ca cerem pozitia elfilor in fabrica
    * deblocheaza zavorul matricei din fabrica
  - *mutaSus* - verifica daca elf se poate muta in sus
  - *mutaJos* - verifica daca un elf se poate deplasa in jos
  - *mutaDreapta* - verifica daca un elf se poate deplasa in dreapta
  - *mutaStanga* - verifica daca un elf se poate deplasa in stanga

- *adaugaElf* - adauga un elf nou creat in fabrica:
  - incuie zavorul listei de elfi
  - daca pozitia elfului nu este luata, adica daca e disponibila, atunci il adauga deja pe lista de elfi si ii cere elfului sa raporteze pozitia sa actuala si deblocheaza zavorul listei de elfi
  - *cerePozitiaElfilor* - cere tuturor elfilor existenti pozitia lor actuala
    - blocheaza zavorul matricei din fabrica
    - blocheaza zavorul listei cu elfi
    - blocheaza zavorul listei cu cadouri
    - toti elfii din lista cu elfi isi raporteaza pozitia lor actuala
    - deblocheaza zavorul matricei din fabrica
    - deblocheaza zavorul listei cu elfi
    - deblocheaza zavorul listei cu cadouri
  - *getCadou* - metoda folosita de un ren pentru a lua un cadou de la fabrica
    - obtine permisul renilor
    - blocheaza zavorul listei de cadouri
    - ia un cadou din lista cu cadouri
    - deblocheaza zavorul listei cu cadouri
    - elibereaza permisul renilor
  - *creeazaCadou* - adauga un cadou in lista de cadouri
    - blocheaza zavorul listei de cadouri
    - pune cadoul in lista cu cadouri
    - deblocheaza zavorul listei cu cadouri
  - *retragereElf* - retrage un elf din fabrica
    - blocheaza zavorul listei cu elfi
    - blocheaza zavorul listei cu fabrici
    - elimina elful din lista si matricea fabricii
    - deblocheaza zavorul listei cu elfi
    - deblocheaza zavorul listei cu fabrici

- **Atelier**
  - *cadouQueue* - mijloc de transfer de cadouri intre Mos Craciun si reni
  - *retragereElfiSemafoare* - un semafor pentru retragerea elfilor
  - *retragereElf* - un thread pentru retragerea elfilor
  - *getElfiCounterLock* - returneaza zavorul elfilor
  - *creeazaFabrici* - creeaza toate fabricile, elfii si renii generati si incepe executia lor

3.5.2 Descrierea fiecarui parametru

- **Elf**
  - *numar* - numarul elfului pentru identificare
  - *X* - coordonata X curenta a elfului in matricea fabricii
  - *Y* - coordonata Y curenta a elfului in matricea fabricii
  - *cadou* - cadoul curent creat de catre elf
  - *fabrica* - fabrica care contine elful
- **SpawnareaElfilor**
  - *fabrica* - fabrica in care thread-ul va spawna elfi
- **TransferulCadourilor**
  - *head* - capul cozii
  - *tail* - coada cozii
  - *cadouri* - numarul de cadouri care trebuie transferate
- **Reni**
  - *numar* - numarul de identificare al renului
  - *fabrici* - toate fabricile existente in atelier
- **FabricaJucarii**
  - *numar* - numarul fabricii
  - *N* - dimensiunea matricei fabricii
  - *elfi* - elfii existenti in fabrica
  - *cadouri* - cadourile existente in fabrica
  - *fabricaLock* - un zavor prin care accesam matricea fabricii
  - *elfiListLock* - un zavor prin care accesam lista elfilor
  - *reniSemafoare* - un semafor pentru maximul de reni permis in fabrica, maximul fiind 10
  - *cadouriLock* - un zavor care permite accesul listei cadourilor
  - *nrExistentElfi* - returneaza numarul de elfi existenti in fabrica
- **Atelier**
  - *nrFabrici* - numarul fabricilor existente
  - *fabrici* - toate fabricile existente

- *spawners* - toate thread-urile generate de elfi existenti
- *nrTotalElfi* - numarul total de elfi existenti
- *elfiCounterLock* - un zavor pentru numarul de elfi existenti
- *reni* - toti renii existenti

3.5.3 Semnificația valorii de return

- **FabricaJucarii**
  - *getFabricaLock* - returneaza zavorul matricei fabrica
  - *getN* - returneaza dimensiunea matricei din fabrica
  - *getNumar* - returneaza numarul fabricii
  - *nrExistentElfi* - returneaza numarul de elfi existenti in fabrica
- **Atelier**
  - *getElfiCounterLock* - returneaza zavorul elfilor
- **Elf**
  - *getNumar* - returneaza numarul de identificare al elfului
  - *getX* - returneaza coordonata X curenta a elfului
  - *getY* - returneaza coordonata Y curenta a elfului
  - *getCadou* - returneaza cadoul curent al elfului

3.6 Implementarea sarcinilor comune si sarcinilor suplimentare

Metode de sincronizare:

- *Sarcini comune*

  Pentru functionarea corecta a fabricilor au fost utilizate 3 zavoare:
  - Un zavor pentru limitarea accesului la matricea fabricii folosita atunci cand un elf se misca in fabrica (2 elfi nu se pot misca in acelasi timp in fabrica), sau cand le cerem elfilor pozitia lor (elfii nu se pot misca in timp ce isi raporteaza pozitia)
  - Un zavor pentru limitarea accesului la lista de elfi, folosita atunci cand un nou elf este adaugat in fabrica (nu pot fi adaugati 2 elfi in acelasi timp, deci pe rand) sau cand se cere pozitia elfilor (lista elfilor nu o putem modifica in acel moment)

– Un zavor pentru limitarea accesului la lista de cadouri, folosita atunci cand cerem pozitia elfilor. Un ren nu poate primi un cadou in timp ce fabrica le cere elfilor sa isi raporteze pozitiile. Mai avem limitarea accesului atunci cand un ren primeste un cadou de la fabrica (unde 2 reni nu pot primi acelasi cadou) si la crearea unui cadou nou in fabrica (adica modificarea listei de cadouri)

Pentru sincronizarea intrarii in fabrica de catre reni am folosit un semafor cu 10 permise pentru fiecare fabrica (o fabrica poate fi accesata de maxim 10 reni in acelasi timp), obtinute atunci cand un ren primeste un cadou din fabrica.

Pentru a avea un numar de identificare pentru fiecare elf, am folosit un zavor pentru accesarea numarului total de elfi, astfel incat sa nu existe 2 elfi cu acelasi numar.

Pentru transferul cadourilor de la reni la Mos Craciun am folosit o coada concurenta, sincronizarea metodelor de adaugare si eliminare a cadourilor in si din coada.

• *Sarcini suplimentare*

Implementarea celor 4 sarcini suplimentare din enuntul problemei, anume retragerea si "odihnirea" unui elf, folosirea clasei Cyclic Barrier si implementarea proprie a barierei:

⋆ ***Retragerea unui Elf***

Pentru a retrage un elf de la fabrica am adaugat un nou thread care va elibera un permis de retragere la fiecare 50 de milisecunde.

Fiecare elf se va muta in fabrica, apoi va incerca sa obtina un permis de retragere din acea fabrica. Retragerea din fabrica inseamna ca un elf va fi eliminat din matricea fabricii si din lista de elfi existenti in fabrica.

⋆ ***"Odihnirea" unui elf - folosind semafoare***

Cand ajunge la diagonala principala a matricei, un elf va incerca sa obtina un semafor pentru a modifica contorul pentru elfii care asteapta la bariera, apoi asteapta cat contorul este $< N$.

Observatie: Numarul maxim al elfilor din fabrica l-am modificat de la $N/2$ la N, deoarece astfel elfii pot atinge diagonala principala.

⋆ ***"Odihnirea" unui elf - folosind Cyclic Barrier***

Aici am rescris programul, adaugand clasa CyclicBarrier, care furnizeaza o metoda mai convenabila de implementare a sincronizarii la bariera.

Cand ajunge la diagonala principala a matricei, un elf va astepta la bariera pana cand cei N elfi vor ajunge la ea, apoi isi va continua actiunile sale - mutarea in fabrica, schimbarea pozitiilor.

⋆ ***"Odihnirea" unui elf - folosind implementarea proprie a barierei***

Pentru implementarea Own Barrier, metoda de asteptare await() va folosi o blocare a contorului pentru elfii care asteapta la bariera, apoi va astepta pana cand contorul este $<$ N.

4 Observatii cu privire la aplicatie

- Pentru ca elfii se odihnesc doar 30 de milisecunde intre crearea cadourilor, deci sunt foarte rapizi, renii trebuie de asemenea sa fie si ei rapizi in a primi cadourile de la fabrici, astfel ca vor dormi intre 10 si 30 de milisecunde.
- Exista mai multi reni care pot oferi cadourile lui Mos Craciun, si de asemenea si Mos Craciun trebuie sa fie rapid in a le primi, astfel ca Mosul nu se odihneste deloc intre primirea cadourilor.
- Daca nu dorim ca un elf sa se retraga imediat dupa ce a creat un cadou (adica dupa ce se muta in fabrica), trebuie sa ii dam thread-ului corespunzator pentru retragere un timp mai indelungat de somn intre eliberarea permiselor de retragere.
- Lista de cadouri a unei fabrici trebuie accesata de un singur ren la un anumit moment, astfel incat semaforul renului se comporta ca o coada de acces pentru a obtine zavorul listei de cadouri.
- Fabricile de jucarii vor fi si ele thread-uri, pentru ca acestea le cer elfilor sa isi raporteze pozitiile la fiecare 3 secunde.
- Toti membrii fabricii care au fost accesati sau modificati din mai multe parti au fost sincronizati, anume acesti membri sunt matricea si listele cu elfi si cadouri.

5 Concluzii...

Ca o concluzie finala, aceasta tema mi s-a parut foarte interesanta. M-am documentat mult pe internet, dar si din cursurile postate pe Classroom. Cel mai mult mi-a placut faptul ca a fost o tema "reala", iar tematica a fost frumoasa.

Spunand aceste lucruri, ma refer la faptul ca tema a fost data inainte de Craciun si cel putin eu am fost surprinsa cand am vazut tematica temei de casa. M-a facut sa vreau sa o citesc si sa o inteleg mai bine, precum sa o si rezolv cum trebuie.

6 Referinte bibliografice

Bibliografie

[1] LaTeX *project site*
- https://www.overleaf.com/learn/how-to/Creating_a_document_in_Overleaf
- https://www.caam.rice.edu/~heinken/latex/symbols.pdf

[2] *Semafoare* https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html

[3] *Zavoare in Java*, https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html

[4] *Cyclic Barrier in Java*, https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CyclicBarrier.html