Sisteme Concurente şi Distribuite

Atelierul lui Moş Crăciun

grupa CR 3.2B, an 3, semestrul I, Calculatoare Română

Niculescu Marius-Andrei

January 23, 2022

1 Proiectarea aplicaţiei experimentale

1.1 Modulele aplicaţiei

Aplicaţia are următoarele module:

- **Elf** este clasa care defineşte atributele unui elf. Fiecare elf rulează pe câte un fir separat.
  - pozitieLinie - indică linia din fabrică pe care se află elful
  - pozitieColoana - indică coloana din fabrică pe care se află elful
  - numarElf - numărul oferit elfului atunci când este creat (folosit pentru identificare)
  - numarCadouriGenerate - numărul de cadouri pe care elful le-a produs atunci când işi termină execuţia
  - fabrica - fabrica la care este asignat să lucreze elful
  - barieraElfi - bariera la care se opreşte elful atunci când ajunge în zona diagonalei principale (când $|pozitieLinie - pozitieColoana| <= 1$)
  - Elf(Fabrica fabrica, int numar, CyclicBarrier barieraElfi) - constructorul care setează fabrica la care lucrează elful, numărul elfului şi bariera
  - void plaseazaElf() - funcţie care generează 2 numere, linia şi coloana din fabrică unde elful încearcă să apară. Dacă acea poziţie este deja ocupată de un alt elf, se generează alte numere pentru o nouă poziţie.
  - void run() - în această funcţie elful va executa mai multe operaţii atâta timp cât fabrica nu a produs numărul de cadouri pe care şi le-a propus. Operaţiile executate de elf sunt: verifică dacă se află în zona diagonalei principale şi în caz afirmativ aşteaptă la barieră, se mută o poziţie în fabrică şi crează un cadou dacă la noua poziţie nu există deja unul, se odihneşte 30 de milisecunde, încearcă să achiziţioneze un permis de la semafor pentru a se retrage, iar dacă fabrica a creat prin intermediul elfilor, numărul de cadouri pe care şi l-a propus, elful afişează numărul de cadouri pe care el insuşi le-a generat, iar apoi "resetează bariera" pentru ca elfii care au rămas blocaţi la aceasta să se trezească, deoarece nu mai trebuie create cadouri.
- **Ren** este clasa care defineşte atributele unui ren. Fiecare ren rulează pe câte un fir separat.
  - numarRen - numărul oferit renului atunci când este creat (folosit pentru identificare)
  - $ArrayList < ArrayList < Integer >> listaCadouriPreluate$ - lista proprie renului pe care o formează atunci când extrage cadouri din fabrici, pentru a fi trimise la Moş Crăciun
  - printwriter - folosit pentru a scrie lui Moş Crăciun cadourile

- listaFabrici[] - renul are acces la toate fabricile din atelier
- terminat - variabilă booleană care constituie condiţia de oprire a buclei din metoda run()
- Ren(Fabrica listaFabrici[], int numarRen, PrintWriter printwriter) - constructorul care setează lista de fabrici din care renul poate să ia cadouri, numărul renului şi printwriter-ul cu ajutorul căruia renul îi scrie lui Moş Crăciun ce cadouri are în listă
- void run() - atâta timp cât variabila booleană *terminat* are valoarea false, renul parcurge lista de fabrici şi extrage câte un cadou. Renii işi termină sarcina de lucru (*terminat* devine true) numai în momentul în care toate fabricile au generat numărul pe care şi l-au propus de cadouri şi au şi lista de cadouri goală (au fost extrase toate cadourile din toate fabricile).

- **RetragereElf** este clasa care se ocupă cu emiterea unui permis către semafor, care poate fi achiziţionat de către elfi pentru a se retrage din fabrică. Obiectul acestei clase rulează pe un fir separat.
  - fabrica - fabrica pentru care funcţionează (fiecare fabrică va avea un obiect de tipul RetragereElf)
  - RetragereElf(Fabrica fabrica) - constructorul care setează fabrica
  - void run() - atâta timp cât fabrica nu a generat numărul de cadouri pe care şi l-a propus, se emite un permis către semaforul din atelier, o dată la o secundă (deoarece elfii apar în fabrică într-un interval de [500,1000] milisecunde, permisele nu trebuie oferite foarte des, deoarece apare riscul de a rămâne fără elfi în fabrică, ceea ce conduce la o buclă infinită, deoarece fabrica nu va avea cum să ajungă la numărul de cadouri propuse)

- **CyclicBarrier** reprezintă propria implementare a unei versiuni simplificate a clasei CyclicBarrier
  - contor - reprezintă numărul de fire care au ajuns la barieră
  - zavorBariera - folosit pentru a proteja accesul la variabila partajată, *contor*
  - numarElemente - reprezintă numărul de fire care trebuie să ajungă la barieră înainte să fie lăsate să continue
  - CyclicBarrier(int numarElemente) - constructor care setează numărul de elemente ce trebuie să ajungă la barieră, pentru a putea fi lăsate să continue
  - void await() - elful care apelează această funcţie încearcă să obţină zăvorul şi în caz de succes incrementează contorul, iar apoi eliberează zăvorul. După aceea, într-o buclă testează dacă toţi elfii au ajuns la barieră. Dacă nu au ajuns, acesta aşteaptă. În momentul în care contorul devine egal cu *numarElemente*, firele îşi reiau execuţia, iar contorul este resetat (este pus la 0).

- void resetare() - această funcţie este apelată atunci când fabrica a generat numărul propus de cadouri. Funcţia setează contorul la $numarElemente$, astfel încât toate firele care aşteaptă la barieră să se trezească, deoarece nu mai trebuie create cadouri şi îşi pot termina execuţia.
- void actualizareNumarElemente() - funcţie care actualizează numărul de fire care trebuie să ajungă la barieră. În cazul în care un elf achiziţionează un permis de la semafor, pentru a se retrage din fabrică, se apelează această funcţie care decrementează valoarea lui $numarElemente$

- **Fabrica** este clasa cea mai importantă din aplicaţie şi ea conţine metodele sincronizate ce asigură o funcţionare cursivă a programului.
  - boolean dispunere[][] - reprezintă dispunerea matricială pe care o are fabrica. Am ales tipul de date boolean deoarece dacă un elf se află într-o locaţie din matrice, atunci o marcăm cu true.
  - listaElfi - lista cu elfii care lucrează la această fabrică
  - listaCadouri - lista cu cadourile care sunt create de elfi atunci când se mişcă în fabrică
  - numarCadouri - contor pentru a ţine evidenţa numărului de cadouri care au fost generate
  - numarCadouriPropuse - numărul de cadouri care trebuie generate pentru ca elfii să îşi termine activitatea
  - Fabrica(int dimensiune, int numarCadouriPropuse) - constructor care setează dimensiunea fabricii şi numărul de cadouri propuse.
  - synchronized void mutaElf(Elf elf) - prima dată se verifică poziţia elfului şi dacă acesta se poate misca în cel puţin una dintre direcţii. Dacă elful este înconjurat de alţii şi nu se poate mişca, acesta aşteaptă până se eliberează o direcţie în care se poate mişca. După ce se asigură că se poate mişca într-o direcţie, acesta marchează vechea poziţie ca fiind liberă (prin punerea valorii false în matricea dispunere), iar apoi îşi actualizează coordonatele conform noii poziţii. La noua poziţie verifică dacă există un cadou în listă în acel loc. Dacă nu există, acesta îl adaugă şi incrementează numărul de cadouri generate în fabrică şi numărul de cadouri generate de către el însuşi. La final el îşi raportează poziţia pentru a pune în matricea dispunere true acolo unde se află el acum.
  - raportareElf(Elf elf) - funcţie care actualizează poziţia în matrice a elfului care o apelează. De asemenea, funcţia apelează notifyAll(); pentru a trezi elfii, care aşteaptă, deoarece sunt înconjuraţi de alţii şi nu au unde să se mute.
  - synchronized $ArrayList < Integer > pregatesteCadouPentruRen()$ - funcţie care extrage primul cadou din listă şi îl returnează renului

care apelează funcţia. Cadoul este stocat în listă sub forma unui $ArrayList < Integer >$ de capacitate doi în care sunt puse află linia şi coloana din matrice unde se află.

- synchronized void retragereElf(Elf elf) - această funcţie identifică elful ce a apelat-o şi setează false în poziţia din matrice unde se află şi îl elimină din lista fabricii

- **Atelier** este clasa care generează datele de test necesare pentru a rula aplicaţia
  - listaReni[] - lista cu renii care preiau cadourile din fabrici
  - listaFabrici[] - lista cu fabricile care generează cadouri cu ajutorul elfilor
  - numarFabrici - numărul de fabrici
  - numarReni - numărul de reni
  - semaforRetrageElf - semaforul pe care îl folosesc elfii în încercarea de a achiziţiona un permis, emis de obiectul clasei RetragereElf, pentru a se retrage din fabrică.
  - retragereElf[] - obiecte ale clasei RetragereElf, fiecare asignat câte unei fabrici
  - writer - este utilizat în constructorul renilor pentru ca aceştia să poată transmită cadourile lui Mos Crăciun
  - bariereElfi - lista de bariere, câte una pentru fiecare fabrică
  - Atelier(PrintWriter writer) - constructor care setează writer-ul
  - void creareFabrici() - funcţie care iniţializează toate obiectele necesare pentru a rula programul. Prima dată se generează numărul de fabrici (între 2 şi 5) şi numărul de reni (între 8 şi 17). Apoi într-o buclă se iniţializează fabricile, unde dimensiunea acestora este un număr aleator între 100 şi 500, iar numărul de cadouri propuse este un număr aleator între 500 şi 800. Odată create fabricile, într-o buclă am dat start la reni pentru ca atunci când elfii încep să creeze cadouri, aceştia să înceapă să le preia. În continuare, pentru fiecare fabrică am asignat un număr aleator de elfi (între 50 şi dimensiunea matricii împărţită la 2), creând şi barierele şi obiectele clasei RetragereElf. După ce am creat numărul de elfi din fabrici, într-o ultimă buclă am iniţializat elfii, i-am plasat în fabrică şi le-am dat start(). Elfii sunt generaţi odată la 500-1000 milisecunde. Din acest motiv am hotărât să dau start la thread-ul care emite permise de retragere, atunci când este creat cel de-al doilea elf.

- **Main** reprezintă punctul de start al aplicaţiei. Aici este creat un socket conectat la numărul de port 9999 pe host-ul "*localhost*". Apoi sunt create OutputStreamWriter şi PrintWriter pentru a putea trimite mesaje pe "canal" către Moş Crăciun. În ultimul rând, este creat atelierul, căruia

i se oferă ca parametru, în constructor, PrintWriter-ul şi este apelată funcţia creareFabrici(). Firul principal va aştepta ca toţi renii să îşi termine execuţia, după care va afişa lista de cadouri a fiecărui ren. La final se închide socket-ul creat deoarece toate cadourile au fost trimise lui Moş Crăciun şi nu mai este nevoie de comunicarea între main şi acesta.

- **MosCraciun** este clasa care se ocupă cu preluarea cadourilor pe care renii le transmit prin "canal". Prima dată se realizează conectarea la portul cu ip-ul 9999 pe care l-am creat în Main, iar apoi creăm un socket care aşteaptă până ce se conectează cineva şi îl acceptă. Odată stabilită conexiunea, MosCraciun citeşte într-o buclă până întâlneşte mesajul null. Apoi închide socket-urile.

1.2 Decizii referitoare la implementare

Metode de sincronizare:

- **Fabrica** Au fost folosite 3 funcţii sincronizate:
  - O funcţie care mută elful în fabrică. Aceasta este sincronizată deoarece 2 elfi nu se pot mişca în acelaşi timp în fabrică deoarece ar apărea probleme de poziţionare şi am avea cazuri în care mai mulţi elfi se găsesc în aceeaşi locaşie.
  - O funcţie care pregăteşte un cadou pentru ren. Aceasta este sincronizată, deoarece elfii nu se pot retrage din fabrică sau nu se pot mişca atunci când renii colectează cadouri
  - O funcţie care retrage un elf din listă. Aceasta este sincronizată deoarece în urma retragerii, elful este şters din listă şi se actualizează matricea, marcând poziţia în care era elful ca fiind liberă. Din acest motiv, ceilalţi elfi nu trebuie să se mişte.

- **RetragereElf** Pentru retragerea elfilor a fost creat un thread separat care emite permise de retragere odată la 1000 de milisecunde (am ales acest timp deoarece elfii apar în fabrică odată la 500-1000 de milisecunde). Atelierul deţine un semafor care primeşte aceste permise, iar elfii de fiecare dată când se mută în fabrică, încearcă să obţină un permis de la semafor.

- **CyclicBarrier** Atunci când un elf ajunge în zona diagonalei principale ($|pozitieLinie - pozitieColoana| <= 1$), acesta aşteaptă la barieră până când toţi elfii din fabrică ajung şi ei în acest punct, apoi îşi continuă activitatea normală de a se muta în fabrică. Pentru implementarea proprie a unei versiuni simplificate a lui CyclicBarrier, atunci când un elf ajunge în zona diagonalei principale, el încearcă să achiziţioneze zăvorul barierei cu ajutorul căruia este capabil să incrementeze contorul, iar apoi aşteaptă atâta timp cât contorul este mai mic decât numărul de fire care trebue să ajungă la barieră.

2 Observaţii

- Din moment ce mai mulţi reni pot să trimită cadouri către Moş Crăciun, acesta nu se odihneşte deloc.
- Elfii se odihnesc câte 30 de milisecunde înainte să se mute pentru a crea alt cadou. Din acest motiv şi renii trebuie să fie rapizi şi să nu se odihnească mult timp.
- Deoarece elfii apar în fabrică la un interval de 500-1000 milisecunde, timpul de repaus al obiectului clasei RetragereElf este de 1000 de milisecunde, pentru a nu genera mai multe permise de retragere decât numărul de elfi din fabrică la momentul respectiv.
- Numai un ren poate accesa lista de cadouri la un moment dat.
- Toate variabilele fabricii, care pot fi modificate de mai multe entităţi, au fost sincronizate (lista de elfi, lista de cadouri, matricea de dispunere).

References

[1] LaTeX project site, `http://latex-project.org/`

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

[3] `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html`

[4] `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CyclicBarrier.html`