**Miklos Balázs**

Tehnici de programare

# Tema2: Cozi

Grupa 30224

Calculatoare

UTCN

## 1. Obiectivul temei

Programul face simularea si evoluarea cozilor unei simulator. Pe parcursul rularii sunt afisate atat in interfata grafica cat si la sistemul de afisare a limbajului Java evoluarea si modificarea cozilor. Datele necesare pot si fi setate de catre utilizatorul prin GUI. Programul si calculeaza tot feluri de timpuri finalizare, medie etc.

## 2. Analiza problemei, modelare, scenarii, cazuri de utilizare

In lumea reala de multe ori intalnim cu obiectul cozii sau cu expresia coada. O coada este un obiect unde se afla regula first-in first-out. Adica primul care a intrat in coada, si iese ca primul din coada. Si in viata de multe ori avem nevoie sa mergem la magazine si sa stam la coada inainte de servire. Timpul cat trebuie sa stam la o coada depinde de numarul clientilor care se alfu inainte de noi si de numarul cozilor. De exemplu daca sunt deschise mai multe coade atunci evident ca mergem la coada care este mai goala decat ceilalti.

Programul aceasta poate fi utilizata pentru simularea cozilor. Numarul cozilor si intervalul de functionare putem sa dam de la interfata de utilizator. Tot asa putem sa stabilim si timpul de ajungere minima si maxima a clientilor respectiv timpul de servire minima si maxima a clientilor. La interfata grafica dupa modificarea datelor din TextFielduri facem click la butonul SET dupa ce facem click la butonul OK si incepe simularea. TextFieldurile o sa fie modificate pe parcursul simularii cu ajungerea si plecarea clientilor la cozi, numarul TextFieldurilor modificate depinde de numarul de cozi functionale. Pe textFieldurile apar ID- ul clientilor.

Idea de baza ese urmatoarea: Este folosit un fir de lucru (Thread) pentru a genera clientii random (cu random arriving time respectiv random service time). Threadul respectiv si adauga timpul de ajungere a clientilor la timpul current. Threadul opreste cand timpul current este mai mare sau egal decat timpul final dat de utilizaator. Am implementat o functie care returneaza ID-ul coadei care este cea mai goala. Clientul este adaugat la coada respectiva. Langa threadul de generarea clientilor mai am cate un thread pentru fiecare coada. Firul de lucru a cozilor intotdeuna doarme atat timp pana un client este servit. Adica fac generarea si servirea clientilor in doua fire de lucru diferite in acelasi timp. Mai am un fir de lucru care am implementat in threadul fiecarei coada si care in fiecare secunde actualizeaza fereastra de GUI. Mai sunt facute niste functii pentru calcularea timpurilor de asteptare medie a clientilor pe coada respectiv pentru calcularea timpului medie-final de asteptare (impart timpurile de medie cu numarul cozilor care a functionat). La fel calzulez si timpul de finalizare a fiecarei coada, de fapt timpul de finaliare este timpul in care este oprit fiecare coada si sunt servite toti clientii. Timpul de finalizare pentru fiecare coada este calculata separat si timpul de finalizare finala va fi maximul timpurilor de finalizare pe coada.

## 3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Proiectul contine doua pachete, un pachet pentru interfata grafica de utilizator si un pachet model. Pachetul de interfata are o singura clasa Window care genereaza fereastra de GUI. Pachetul de model contine 5 clase: App, Client, Coada, RandomClientGenerator, Simulator. In clasa App este facuta main care apeleaza new Simulator(). De fapt in clasa Simulator face inceperea, pornirea threadurilor si a programului. In clasa simulator este apelat si crearea ferestrei new Window().

Clasa Client contine informatii despre un client. Ca cariabila de instanta este facut un ID, un timp de ajungere (arriveTime) si un simp necesar de servire a clientului (serviceTime). Clasa Coada am mai multe variabila de instanta pentru a sincroniza threadurile care ruleza in acelasi timp (boolean wasStarted, int x, boolean running). Variabila wasStarted devine adevarat cand thredul de actualizare a rezultatelor (afisare) a fost pornita cu metoda .start(). De aceea am nevoie de variabila aceasta ca threadul respectiv e apelat in interiorul thredului de coada si ca sa nu fie pornita de mai multe ori (evitarea erorei). Variabila x devine 1 doar daca in coada respectiva a ajuns vreodata un client si variabila running este folosit pentru aflarea starei a thredului de coada. Mai am aici o lista de clienti, un ID al coadei, un timp de finalizare a coadei (finishTime) si un timp de asteptare de medie a clientilor in coada (avarageWaitingTime). Mai este facuta o metoda CalculateWaitingTime() care calculeaza timpul de asteptare a unui client la coada.

In clasa RandomClientGenerator am ca variabile de instanta un timp de ajungere minima si maxima a clientilor, un timp de servire minima si maxima a clientilor si un ID a cloentilor care tot o sa fie incrementat cu 1 ca toti clienti sa aiba ID diferit. Mai am un Random nr care de fapt fa genera numere aleatoare. Generez un numar aleatoriu pentru arriveTime si serviceTime intre valorile de limita. Clasa mai are o metoda generateRandomClient() care de fapt genereaza un client cu timpi random.

In final in clasa Simulator tot am niste variabile de instanta pentru sincronizarea firelor de lucru, adica booleana running care indica starea threadului de generare a clientilor. Mai am variabilele curentTimp si maxTimp cu care masor timpul, cu ajutorul lor opresc thredurile. Main am un random client generator r si o variabila number of queuees. Cea mai importanta variabila de instata este lista cozilor (LinkedList queues). Ca metode in aceasta clase este implementata metoda getFinishTime(), getAvgTime() si returnIDOfMostGoalQueue(). Metoda getFinishTime calculeaza timpul de finalizare a simularii, getAvgTime calculeaza timpul de asteptare de media dintre toti clienti si metoda returnIDOFMostGoalQueue returneaza ID-ul coadei care este cea mai goala.

In final in pachetul de interfata clasa Window contine niste variabile de instanta pentru a crea fereastra de GUI. Mai ales am doua butoane statice si 6 textFielduri tot statice ca sa pot sa accesez din celalalte clase. Mai am metodele createTextFields() si addThingsToPanel() cu care creez fieldurile de text si adaug ei cu butoanele SET si OK. Butonul SET o sa seteze valorile min/mam arrive/service time si number of clients respectiv maxtimp la valori care scriem noi la textfields. Butonul OK porneste thredurile. De fapt toata fereastra este o singura panela declarat cu JPanel mainPanel, care este adaugat la fereastra frame care este un JFrame.

Langa chetile enumerate evident ca fiecare clasa au constructoarele proprii, respectiv gettere si settere. Toate variabile de instanta este declarat cu private adica proiectul respecta si regula incapsularii. Sunt niste exceptii la variabile de instante cu ajutorul carora sincronizez threadurile fiinda in alt mod nu mai puteam accesa.

## 4. Implementare

Pachetul model

Clasa **App** este clasa principala de unde este apelat functia new Simuator() care incepe sa ruleze thredurile si fereastra.

-public static void main(String args[]) returneaza void.

Clasa **Client** are variabilele de instanta int ID, int ServiceTime si int ArrivalTime. Rolurile lor au fost explicate anterior.

-public Client()- constructorul creaza un client.

-public int getID()- returneaza ID.

-public void setID(int iD) -returneza void, seteaza ID.

-public int getServiceTime()- returneaza serviceTime.

-public void setServiceTime(int serviceTime)- returneaza void, seteaza serviceTime.

-public int getArrivalTime()- returneaza arrivalTime.

-public void setArrivalTime(int arrivalTime)- returneaza void, seteaza arrivalTime.

Clasa **Coada** extinde Thread, au variabilele de instanta boolean wasStarted, LinkedList <Client> Coada, boolean running, int ID, int finishTime, float avarageWaitingTime.

-public Coada(int ID)- constructorul, creaza o coada.

-public void runInteriorThread()- returneaza void, metoda thredului care actualizeaza in fiecare secunda fereastra de GUI.

-public void run()- returneaza void, metoda threadului cozii, ruleaza pana cand coada devine goala si pana cand clasa Simulator nu mai genereaza nici un client.

-public int calculateWaitingTime(Client client)- returneaza un int, calculeaza timpul de asteptare a unui client in felul urmator: adauga la timpul de service a clientului toata timpurile de service al celorlalti clienti care stau in fata clientului nostru.

-public LinkedList<Client> getCoada()- returneaza LinkedList, returneaza coada de clienti.

-public int getID()- reutrneaza int, ID-ul coadei.

-public int getFinishTime()- returneaza un int, timpul de finalizare a coadei.

-public float getAvarageWaitingTime()- returneaza timpul de asteptare medie a coadei.

Clasa **RandomClientGenerator** au variabile de instanta Random nr, int minArrivingTime, int maxArrivingTime, int minServiceTime, int maxServiceTime, int id.

-public Client generateRandomClient()- retuneaza Client, clientul generat cu timpul de ajungere intre min si maxArrivingTime, cu timpul de servire intre min si maxServiceTime si cu un ID unic.

-public void setMinArrivingTime(int minArrivingTime)- returneaza void, seteaza minArrivingTime.

-public void setMaxArrivingTime(int maxArrivingTime)-retuneaza void, seteaza maxArrivingTime.

-public void setMinServiceTime(int minServiceTime)- returneaza void, seteaza minServceTime.

-public void setMaxServiceTime(int maxServiceTime)- returneaza void, seteaza maxServiceTime.

Clasa **Simulator** extinde Thread, au variabilele de instanta int maxTimp, int currentTimp, int nrOfQueues, boolean running, RandomCleintGenerator r si LinkedList<Coada> queues.

-public Simulator()- constructorul, creaza un nou fereastra, un nou RandomCleintGenerator si adauga cate un ActionListener pentru butoanele OK si SET. SET seteaza valorile necesare la valorile care introducem in textFielduri iar OK porneste simularea.

-public void run()- returneaza void, metoda de rulare a threadului de generare a clientilor. Metoda ruleaza pana cand currentTimp mai mare sau egal decat maxTimp. La finalul executiei apare o fereastra despre timpul de finalizare si timpul de asteptare de medie a cleintilor.

-public int returnIDOfMostGoalQueue()- returneaza int, ID-ul coadei care este cea mai goala si unde vor fi adaugati clientii noi.

-public float getAvgTime()- returneaza float, calculeaza timpul de asteptare medie a tuturor clientilor care a participat la vre-o coada.

Pachetul interfata

Clasa **Window** contine variabila de instanta JPanel mainPanel si variabilele statice public static JtextField[] textFields, JButton OK si JButton SET. La vectorul textFields avem 6 textfiled-uri, numarul fiecarei textfield coreaspunda numarul de ID a cozilor functionale. Cand pornim programul aici introducem datele necesare, setam cu butonul SET si rulam programul cu butonul OK.

-public Window()- constructorul creaza un nou JFrame unde adauge textfield-urile si butoanele apeland urmatoare metode de ajutatoare:

-public void createTextFields()- returneaza void, creaza cele 6 textfielduri.

-public void addThingsToPanel- adauga la variabila mainPanel toate butoanele si textfield-urile create anterior.

## 5. Rezultate

Programul simuleaza evaluarea cozilor, actualizeaza in fiecare secunda starea cozilor functionale. Programul este user friendly fiindca este implementat si o unterfata grafica de utilizator cu ajutorul caruia este mai usoara simularea setarea datelor necesare pentru simulare a cozilor (setarea timpului de ajungere minima si maxima a clientilor, setarea timpului de servire minima si maxima a cleintilor, numarul cozilor si in final timpul maxim de simulare), vederea rezultatelor.

Langa aceasta mai apare rezultatele si pe sistemul de afisare a limbajului de programare Java adica pe Consol, acolo este afisat fiecare client cu timpul lui de ajungere respectiv starea cleintului si ID-ul coadei unde a fost servit clientul. Afisarea se face pentru toti clienti, este actulazizat nu doar in fiecare secunda ca si actualizarea GUI, ci in fiecare moment cand ajunge un nou client sau cand clientul termina la o coada.

Simularea funcioneaza corect fiindca daca introducem acelasi date si facem simularea de la o singura coada pana maxim 6 cozi, putem observa ca timpul de finalizare scare nu marirea numarului de cozi. Tot asa daca setam la timpul de servire sau timpul de asteptare a clientilor o valoare mai mica, simularea tot se termina mai devreme decat in cazul contrar.

## 6. Concluzii

Programul care am implementat poate fi foarte folositor fiindca si in lumea reala intalnim de mai multe ori cu cozi, cand suntem intr-un magazin sau cand trebuie sa asteptam la un orice fel de rand. Cand randul este mare si este doar o coada de servire evident ca trebuie sa asteptam mai mult decat in caz contrar.

Simularea poate foloseasca oricine de la o persoana individuala pana la o firma mai mare unde este nevoie de scaderea si micsorarea cozilor. Evident ca profitul unui magazin de exemplu Kaufland sau Lidl creste daca clintii care merg acolo sa cumpere ceva vor fi servite mai devreme. Exista mai multe situatii, dar programul aceasta poate sa simuleze aproape orice fel de situatie de evoluare a cozilor. Tot asa poate fi modificat usor de exemplu ca nu sa functioneze doar pentru 6 cozi ci sa funcioneze pentru orice numar de coada (modificarea se face mai ales la adaugarea mai multe textfielduri la GUI).

## 7. Bibliografie


-http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

-http://www.tutorialspoint.com/java/util/timer_schedule_period.htm

-http://javahash.com/java-concurrency-future-callable-executor-example

## 8. Diagrama UML