TEHNICI DE PROGRAMARE

Tema 2 :

**SISTEM DE IMPLEMENTARE AL COZILOR**

Nume : Toderean

Prenume : Alexandru

An : II

Seria : A

Grupa : 30224

Profesor Coordonator : Dorin Moldovan

CUPRINS

1. Obiectivul temei . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pag 2-3
2. Analiza problemei . . . . . . . . . . . . . . . . . . . . . . . . . . . .pag 3
3. Proiectare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pag 5
4. Implementare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pag 6
5. Rezultate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pag 6
6. Concluzii, Dezvoltări ulterioare . . . . . . . . . . . . .pag 7-8
7. Bibliografie . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pag 8

**1.Obiectivul temei**

Obiectivul acestei teme este de a realiza un program care simulează comportamentul unui magazin. Mai exact, această aplicație va implementa un algoritm care să ajute magazinul să funcționeze cât mai eficient. Astfel, clienții din magazin care si-au terminat cumpărăturile și doresc să plătească pentru a putea pleca, vor fi îndreptați spre coada cea mai scurtă cu ajutorul acestui program. Cu alte cuvinte, acest program implementează o problemă din lumea reală, tipică pentru programarea orientată pe obiect. Dacă toate magazinele ar avea o astfel de aplicație, cu siguranță timpul de așteptare ar fi mult mai mic, totul s-ar petrece mult mai rapid, ceea ce ar mulțumi clienții într-un mod deosebit.

Din punct de vedere al cunoștințelor, prin acest proiect se urmărește înțelegerea threadurilor în Java . Totododată, pentru o mai bună vizualizare și înțelegere al acestui proiect , a fost implementată o interfață grafică prin care utilizatorul poate comunica mult mai clar cu programul.

**2. Analiza problemei**

Pentru realizarea acesei teme avem nevoie de o bună înțelegere a concurenței în limbajele de programare orientată pe obiect. Concurența este abilitatea de a rula mai multe părți ale unui program sau mai multe programe în paralel. Dacă sarcini consumatoare de timp pot fi realizate asincron sau în paralel, acest lucru poate îmbunătăți debitul și interactivitatea programului.

Pentru a implementa corect aplicații de acest gen, cea mai bună soluție rămâne utilizarea threadurilor. Conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către [sistemul de operare](http://ro.wikipedia.org/wiki/Sistem_de_operare). Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod [în paralel](http://ro.wikipedia.org/wiki/Calcul_paralel) în interiorul aceluiași [proces](http://ro.wikipedia.org/w/index.php?title=Proces_(sisteme_de_operare)&action=edit&redlink=1). Câteodata însă, aceste portiuni de cod care constituie corpul threadurilor, nu sunt complet independente și în anumite momente ale execuției, se poate întampla ca un thread să trebuiască să aștepte execuția unor instructiuni din alt thread, pentru a putea continua execuția propriilor instrucțiuni. Această tehnică prin care un thread asteaptă execuția altor threaduri înainte de a continua propria execuție, se numește sincronizarea threadurilor.

Astfel, eu am ales să folosesc câte un thread pentru fiecare coadă ( de la o casă de marcat ). Când începe simularea magazinului, threadurile se pornesc și așteaptă clienții. După ce clienții trec de casa de marcat, threadurile se vor închide. Datorită faptului că nu avem un număr fix de clienți sau de cozi, am utilizat ArrayList. Modelarea în sine nu a fost grea dar nici ușoară. Chiar dacă codul nu este unul bogat, pentru a implementa algoritmul acestei aplicații cu threaduri, trebuie analizată foarte bine problema. Pe lânga threadurile folosite pentru fiecare coadă în parte, am mai folosit un thread principal pentru inserarea în coada cea mai scurtă, clientul care a ajuns la coadă ( adică are timpul de sosire egal cu contorul ). În momentul în care toți clienții din magazin au fost serviți și practic ies afara din magazin, acest thread se termină și va transmite un semnal celorlalte thread-uri care la rândul lor se vor încheia și ele.

Cu ajutorul interfetei, putem observa functionalitatea si eficienta thread-urilor.

Această analiză simplistă a problemei a dus la modelarea paradigmelor OOP care în final oferă o rezolvare elegantă a problemei, trecând testele și în diferite cazuri critice de utilizare.

Programarea orientată pe obiecte ( OOP ) este o paradigmă de programare, axată pe ideea încapsulării, adică grupării datelor și codului care operează asupra lor, într-o singură structură. Un alt concept important asociat programăii orientate obiect este polimorfismul, care permite abstractizări. Acestea la rândul lor, permit o descriere conceptuală mai simplă a soluției.

**3.Proiectare**

Proiectarea propriu-zisă a avut loc după proiectarea modelului UML, cel din urmă constând într-o diagramă. Cu ajutorul acestei diagrame UML se pot evidenția legăturile dintre clase, pachete, respectându-se altfel conceptul OOP.

UML este un limbaj vizual de modelare, el nu este încă un limbaj de programare, deoarece nu dispune de întreg sprijinul semantic și vizual pentru a înlocui limbajele de programare. Limbajul este destinat vizualizării, specificării, construirii și documentării sistemelor de aplicații, dar are limitări în ceea ce privește generarea codului. UML reunește cele mai bune tehnici și practici din domeniul ingineriei progamării, care și-au dovedit eficiența în construirea sistemelor complexe.

Diagramele sunt grafuri care prezintă simboluri ale elementelor de modelare (model element) aranjate astfel încât să ilustreze o anumită parte sau un anumit aspect al sistemului. Un model are de obicei mai multe diagrame de același tip. O digramă este o parte a unui view specific, dar există posibilitatea ca o diagramă să facă parte din mai multe view-uri, în funcție de conținutul ei.

Pentru implementarea programului s-a creat un pachet în care sunt incluse toate clasele. După cum se poate observa din diagrama UML de mai jos, în acest proiect, au fost implementate 7clase si o interfata : Client, Server, ConcreteStrategieQueue,ConcreteStrategieTime,Scheduler,Server,SimulationFrame,SimulationManager,Strategy(interfata). Clasa reprezintă un tip de date obținut prin abstractizarea unor obiecte similare. O clasă este modelul unui obiect. Clasa spune mașinii virtuale să construiasca un obiect de tipul respectiv. Din punct de vedere structural, o clasă este descrisă prin denumire, atribute (date), metode (operatii) .

.

Clasa Client definește practic un obiect de tipul clientului. În fiecare server vom avea unul sau mai mulți clienti și pentru asta am definit în clasa Server o listă de clienți ( ArrayList<Client> ). Din acest motiv, între aceste două clase există o relație de agregare. Numită și compunere, agregarea reprezintă pur și simplu prezența unei referințe la un obiect într-o altă clasă. Acea clasă practic va refolosi codul din clasa corespunzătoare obiectului.

Această aplicație a trebuit contruită astfel încât să poată realiza simultan mai multe activități. Firele de execuție reprezintă mecanismul prin care pot fi implementate în cadru unui program secvențe de cod ce se execută virtual în paralel. Limbajul Java nu peremite moștenire multiplă. Aceasta înseamnă că dacă avem o clasă care moștenește deja o clasă, și dorim să o transformăm în fir de excuție nu vom putea folosi metoda de extindere a clasei Thread. Pentru a putea rezolva astfel de situații Java pune la dispoziție o altă metodă de construire a firelor de execuție folosind interfața Runnable. În aplicația noastră, această interfață a fost implementată atât în clasa SimulationManager.

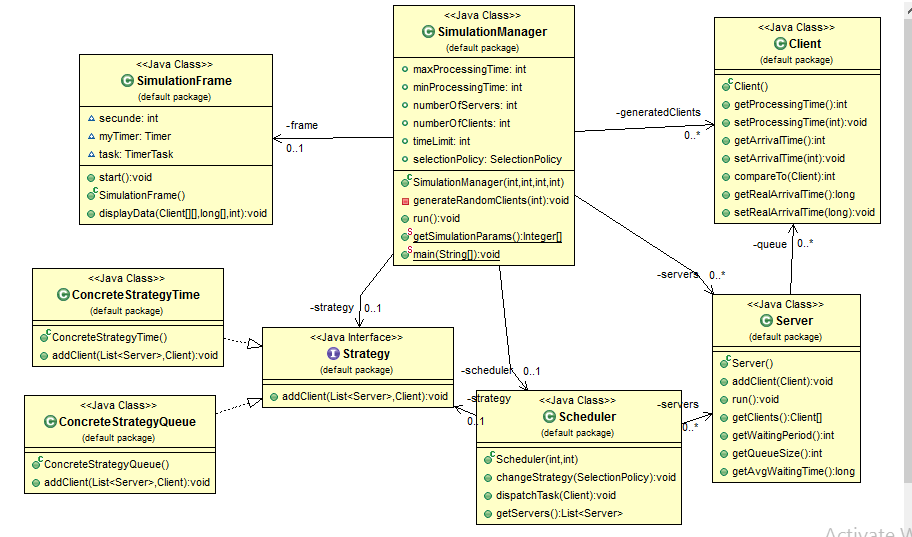
Clasa SimulationManager are rolul de a realiza ComboBox-ul adica caseta de dialog in care vom pune parametrii de creare a cozilor,dar si totodata generarea aleatoare a clientilor si implementarea cozilor.

**4. Implementare**

Implementarea, codul propriu-zis a fost scris după modelul UML văzut mai sus. Se vor descrie acum în detaliu clasele, metodele utilizate pentru realizarea acestui program.

Clasa Client definește practic un obiect de tipul clientului având următoarele atribute: *arrivalTime(timpul de sosire,care reprezinta momentul in care clientul soseste la coada)*, *processingTime*(timpul de procesare, reprezintă durata de timp necesară servirii clientului). În această clasă va fi implementată doar gettere si settere si o metoda de comparare a timpilor de sosire.

În clasa SimulationManager se va implementa interfața Runnable. Definirea interfețtei Runnable a fost de fapt necesară pentru execuția ca fir de execuție a codului conținut în obiecte care nu reprezintă extensii ale clasei Thread. Un astfel de obiect ar trebui să moștenească metode din două origini. Deorece Java nu admite moștenire multiplă problema se rezolvă extinzând o clasă și implementând în același timp o interfață. Atributele clasei Server sunt: *queue*( un LinkedBlockingQueue<Client> ) care reprezintă lista clienților din coadă, și o variabila waitingPeriod care este declarata AtomicInteger. În constructorul acestei clase se va instantia lista de



clienti si perioada de asteptare. În clasa Server avem gettere și settere deoarece cozile se reprezintă ca niște memorii FIFO ( First In, First Out . Metoda *public void addClient()*  imi adauga clientii in coada.. Cea mai importană metodă din această clasă este metoda *public void run()* care trebuie suprascrisă deoarece am clasa Server implementează interfața Runnable. Practic prin această metodă se descrie comportamentul pentru ficare coadă din magazin. Fiecare thread care se va executa pe fiecare coadă va implementa acestă metodă. Prin acestă metodă se va afișa la fiecare secundă coada și în același timp se va decrementa timpul de așteptare pentru fiecare client. Apoi fiecare client va fi scos din coadă.

Acest thread va rula pana cand se va termina timpul maxim dat in dialogBox . Acesta se verifica în fiecare moment cu timpul de servire generat pentru fiecare client. Dacă cele două sunt egale, clientul cu timpul de servire respectiv, va fi adăugat în cea mai scurtă coadă, va fi servit iar apoi va fi scos afară din coadă.

**5. Rezultate**

Rezultatele acestei aplicații pot fi osbervate clar rulând aplicația, pentru că doar în felul acesta se vede cum clienții intră în cozi (alegând tot timpul coada cu timpul de așteptare cel mai scurt), așteaptă să fie serviti în tot timpul de așteptare, iar apoi sunt scoși din coadă.

Rezultatele variaza in functie de timpul dat in interfata,de numarul de clienti si numarul de case.

**6.Concluzii și dezvoltări ulterioare**

În zilele noastre timpul este foarte important și oricât de bine ne-am organiza, un pic de timp în plus nu strică niciodată. Este mult mai bine să ne mișcăm eficient sau să avem un program care să ne ajute în această privință . De aceea, consider că această aplicație poate fi de mare ajutor atât angajaților unui supermarket cât și clienților care merg să își facă cumpărăturile acolo. Prin introducerea unei astfel de sistem într-un magazin mare s-ar reduce timpul de așteptare pentru a putea fi servit, plus că toate lucrurile ar merge într-un ritm mult mai alert și astfel am economisi timp considerabil.

În concluzie, consider că această temă ajută la înțelegerea clară a thread-urilor, acestea oferind cea mai corectă metodă de a implementa un astfel de program, respectând totodată conceptele programarii orientate pe obiect.

Firele de execuție ( threads ) reperezină porțiuni de cod ale aceluiași program care se pot executa în paralel una fața de cealaltă. Cu ajutorul lor, limbajul Java suportă programarea concurentă.

Ca oricare alt program, bineînțeles că și acesta poate fi dezvoltat ulterior din mai multe privințe. De exemplu, aș putea crea niște butoane prin care să deschid sau să închid casele în timp real, sau de ce nu, sa se închidă și să se deschidă automat în funcție de numărul de clienți aflați în magazin.

După realizarea acestui proiect, am învățat cum să creez și să lucrez cu o interfață swing și totodată mi-am dezvoltat abilitățile în programarea orientată pe obiect ( OOP ) .

Programarea orientată pe obiect este unul din cei mai importanți pași făcuți în evoluția limbajelor de programare spre o mai puternică abstractizare în implementarea programelor. Ea a apărut din necesitatea exprimării problemei într-un mod mai natural ființei umane. Astfel, unitățile care alcătuiesc un program se aproprie mai multde modul nostru de a gândi decât modul de lucru al calculatorului.

**7.Bibliografie**

1. Java de la 0 la expert ( ediția a II – a ), Autor (i) : Ștefan Tănasa, Cristian Olaru, Ștefan Andrei, Editura : Polirom, An apariție : 2007
2. Introducere în universul JAVA, Autor (i) : Horia Georgescu, Editura : Tehnica
3. <http://en.wikipedia.org/wiki/Swing_%28Java%29>
4. <http://stackoverflow.com>
5. <http://docs.oracle.com/javase/tutorial/uiswing/>
6. <https://www.draw.io/>