****

**DOCUMENTAȚIE TEMA 2**

**QUEUES SIMULATOR**

**SISTEM DE MINIMIZARE A TIMPULUI DE AȘTEPTARE LA COZI A UNUI CLIENT**

**Trifu Diana-Maria**

**Grupa 30223**

**Profesor Laborator: Dorin Moldovan**

Cuprins:

**1. Cerințe funcționale …………………………………………………………………….……. 3**

**2. Obiectivul temei ………………………………………………………………………..……. ..3**

**2.1 Obiective principale …………………………………………………...……...…...….…3**

**2.2 Obiective secundare ………….……………………………………………………....… 4**

**3. Analiza problemei ……………………………………………….………………………....… 4**

**3.1 Scenarii …………………………………………………………………………..…..…4**

**3.2 Modelare...………………………………………………………………………..….....5**

**4. Proiectarea ……………….………………………………………………………….………….. 5**

**4.1 Structuri de date folosite ………………………………………….…………..........…5**

**4.2 Diagrama de clase....………….………………………………………………………….6**

**4.3 Algoritmi ………………………………………………………………..………………..…6**

**5.Implementare …….…………………………………………………………...……………….....7**

**6. Rezultate …………..………………………………………………………...………………….....8**

**7.Conluzii ……………………………………………………………………...……………..………8**

**8.Bibliografie …………………………………………………...………………………...………....9**

1. **Cerințe funcționale**

**Propuneți, proiectați și implementați o aplicație care are scopul de a analiza timpul de așteptare al clienților la o coadă pentru a-l determina și a-l minimiza. Următoarele date ar trebui luate în cosiderare ca și date de intrare la citirea dintr-un fișier text pentru aplicația care va fi construită:**

* **Numărul de clienți (N)**
* **Numărul de cozi (Q)**
* **Intervalul de simulare (tMaxSimulation)**
* **Minimul și maximul timpului de sosire (tMinArrival < tArrival < tMaxArrival)**
* **Minimul și maximul timpul de procesare în cadrul cozii (tMinService < tService <tMaxService)**

**Datele de ieșire ale aplicației vor fi scrise într-un fișier text care va conține detaliile (statusurile) din timpul simulării, dar și timpul mediu de așteptare al clienților care au participat în procesul de simulare.**

**Considerații de implementare:**

* **Folosirea limbajului de programare Java**
* **Implementarea claselor cu maxim 300 de linii de cod**
* **Implementarea metodelor cu maxim 30 de linii de cod**
* **Generarea random a clineților care iau parte în procesul de simulare**
* **Structurile folosite sa asigure siguranța threadurilor**
* **Generarea unui fișie .jar pentru a asigura rularea aplicației la accesare**
* **Cozile folosite ar trebui să se închidă și să se deschidă dinamic. Inițial toate cozile vor fi închise, iar după ce clienții vor fi distribuiți cozilor, acestea vor devein deschise și se vor închide doar după ce coada devine goala**

1. **Obiectivul temei**
   1. **Obiectivele principale**

**Obiectivul principal al proiectului este de a creea o aplicație care să pună la dispoziție utilizatorului un sistem de simulare minimizat care calculează timpul mediu de așteptare al unor client la cozi. Cozile sunt folosite în viața reală pentru a modela diverse domenii ale lumii. În contextual programării, o coadă este o colecție în care entitățiile sunt păstrate în ordine. Operațiile de bază care se pot executa pe această structură de date sunt: enqueue și dequeue. Ordinea de păstrare este FIRST IN FIRST OUT(FIFO). Într-o structură de date de tip FIFO, primul element care a fost adăugat va fi totodată și primul care va fi șters. Un bun exemplu de coadă din viața reală este o coadă de consumatori într-un magazine, care așteaptă să își achite produsele. Primul client care se va pune la coadă va fi totodată și cel care va pleca primul.**

* 1. **Obiectivele secundare**

**Obiectivele secundare ale temei reprezintă pașii careau fost urmați pentru atingerea obiectivului final.**

1. **THREADS**

**În contextual programării, conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către**[**sistemul de operare**](https://ro.wikipedia.org/wiki/Sistem_de_operare)**. Este folosit pentru a eficientiza execuția programelor, executând porțiuni distincte de cod**[**în paralel**](https://ro.wikipedia.org/wiki/Calcul_paralel)**în interiorul aceluiași**[**proces**](https://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. Threadurile sunt diferite față de clasicele procese gestionte de sistemele de operare ce suportă multitasking, în principal prin faptul că, spre deosebire de procese, toate threadurile asociate unui proces folosesc același spațiu de adresare. Procesele sunt în general independente, în timp ce mai multe threaduri pot fi asociate unui unic proces. Procesele stochează un număr semnificativ de informații de stare, în timp ce threadurile dintr-un proces impart aceeași stare, memorie sau alte resurse. Procesele pot interacționa numai prin mecanisme de comunicare interproces speciale oferite de sistemul de operare (semnale, semafoare, cozi de mesaje și altele asemenea). Cum împart același spațiu de adresare, threadurile pot comunica prin modificarea unor variabile asociate procesului și se pot sincroniza prin mecanismele proprii. În general este mult mai simplu și rapid schimbul de informații intre threaduri decât între procese.**

1. **Aplicația dezvoltată va analiza cozile folosindu-ne de threaduri. Aceasta aplicație poate fi vazută ca un magazine cu un număr maxim de case de marcat, un număr maxim de clienți care vor avea ca limite: un timp limită de sosire la coadă și un timp limită de stat în cadrul cozii. Aceste date de intrare vor fi introduce de catre utilizator într-un fișier text, iar detaliile simulării și timpul mediu de așteptare calculate vor fi primate de către utilizator tot într-un fișier text generat automat de către aplicație.**
2. **Analiza Problemei**

**Programarea orientate peobiect oferă posibilitatea începerii dezvoltării unui proiect fără nevoia efectivă de a cunoaște de la început toate amănuntele funcționării effective a aplicației care urmează să fie dezvoltată. Această strategie folosită este cunoscută sub numele de “Top Down.” O abordare de acest tip este esențială cîntr-un proiect de acest tip, deoarece îți oferă o perspectivă mult mai bună asupra sub-sistemelor care alcătuiesc întregul. O astfel de metodă oferă posibilitatea proiectantului să își faca o imagine de ansamblu asupra sistemului, fără a fi nevoit să detalize fiecare nivel al subsistemelor.**

* 1. **Scenarii**

**Descrierea funcționării aplicației și a modului de introducere a datelor în fișierul text de intrare:**

**Exemplu de date de intrare pentru fișierul text:**

**Numărul de clienți: 10**

**Numărul de cozi: 4**

**Timpul minim de sosire la coadă: 0**

**Timpul maxim de sosire la coadă: 26**

**Timpu minim de procesare: 4**

**Timpul maxim de procesare: 7**

**Timpul de simulare: 40**

**Aceste date vor fi introduce în fișierul text de intrare astfel:**

**10**

**4**

**40**

**0, 26**

**4,7**

**Introducerea sub o alta forma a datelor de intrare vor duce la eșecul simulării, astfel încăt citirea datelor de intrare din fișier nu va mai fi realizată corespunzător.**

**Posibilele scenarii de eșec care pot apărea: introducerea sub o formă greșită a datelor în fișier, datele de intrare din fișier nu sunt numere. Simularea se va termina în momentul în care timpul este egal cu timpul de simulare. Fișierul de ieșire: statusul cozilor în fiecare secundă din timpul simulării + timpul mediu de așteptare al clienților la coadă.**

**3.2 Modelare**

**Utilizitarolu va putea accesa aplicația fie prin intermediul Eclipse, fie prin intermediul fișierului cu extensia .jar, care ofera o compilare imediată a programului printr-unsimplu click. Pentru a modifica datele de intrare se va accesa fișierul text dat ca parametru în cod. Dupa o perioada egala cu timpul de simulare introdus în fișierul text, în fișierul dat ca paramtru de ieșire se va afla o detaliere a procesului simulat, dar și timpul mediu de așteptare la coadă a unui client. Clienții sunt generate random prin intermediul unei metode, pentru ca rezultatul obținut să fie căt mai obiectiv.**

1. **Proiectare**

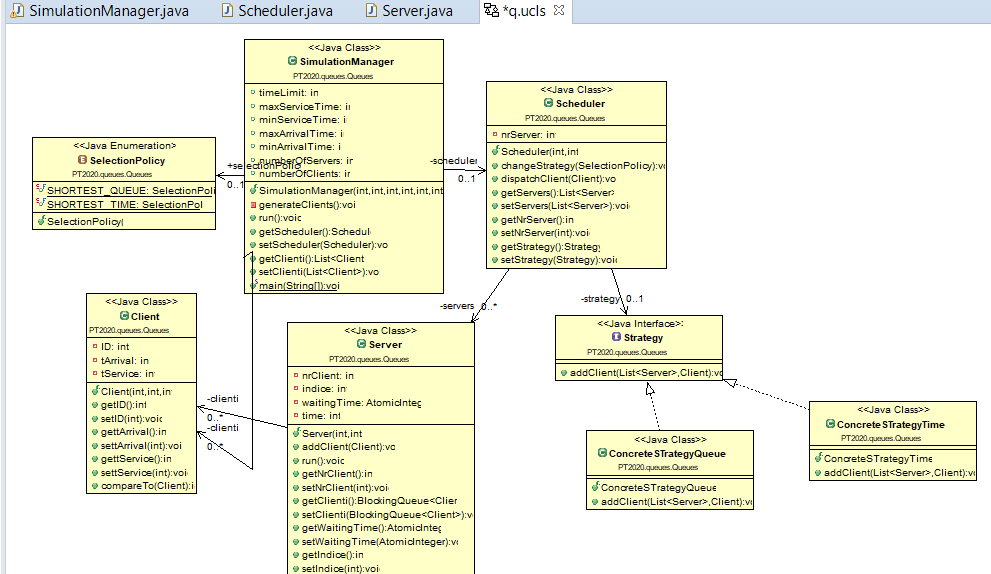
**4.1. Structuri de date și clasele folosite**

**Ca și clase principale folosite sunt: Client, Server, Scheduler și SimulationManager. Clasa Client descrie persoanele care vor fi distrubuite în cozi, clineții magazinului, clasa Server descrie cozile în care sunt disribuițiclienții, cozile de la casele de marcat ale unui magazin, clasa Scheduler realizeaza legătura între clasele precedente. Această clasă se va ocupa de stabilirea strategiei de distribuire a clienților în cozi. Clasa SimulationManager se va ocupa de distribuirea în cozi a clienților, ținând cont de strategia stabilită in clasa Scheduler. Clasele ConcreteSTrategyQueue și ConcreteSTrategyTime implementează interfața Strategy, acestea împreună cu clasa Scheduler alcătuind un Strategy Pattern (policy pattern). Strategiile luate în considerare de către aplicație sunt: distriubuirea cliențiol la cozi în funcție de timpul minim sau în funcșie de numărul minim de persoane al unei cozi. Strategia folosita de aplicație este cea care elege coada în care va fi distribuit clientul în funcșie de timpl minim de așteptare.**

**Ca și structuri de date folosite am ales ArrayList-urile și LinkedBlockingQueues. ArrayListurile reprizintă o structură de date ușor de utilizat datorită numeroaselor etode care sunt deja gata implementate, ceea ce oferă o mobilitate foarte mare și ușurință în lucrul cu acestea. LinkedBlockingQueues au fost utilizate pentru reprezentarea cozilor de clienți, acestea fiind capabile să ofere siguranță mai mare în lucrul cu threadurile.**

**4.2. Diagrama de clase**

**Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații software. Diagrama de clase UML este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxionomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.**

**4.3 Algoritmi**

**Server – Runnable:**

**Modele folosite:**

* **Clienți(BlockingQueue<Client>… : LinkedBlockingQueue)**
* **WaitingPeriod (AtomicInteger): decrementată de threadul curent, odata ce un Client și-a terminat perioada de timp de procesare în cadrul cozii și incrementată de Scheduler atunci când sunt adăugați noi clienți cozii.**

**Scheduler**

**Trimite clienții în cozi în funcție de strategia stabilită**

**Modele folosite:**

* **Servere**
* **Constrângeri: maxNoServers, Strategy**

**Scheduler – Strategy Pattern**

* **Se alege politica de distribuire a clienților în cozi**

Strategy

Scheduler

ConcreteSTrategyTime

ConcreteSTrategyQueue

**SimulationManager – Runnable:**

* **Generarea Random a clienților cu: timp de sosire, timp de procesare**
* **Bucla de simulare: update pentru timpul curent, apelarea Scheduler-ului pentru distribuirea clienților în cozi**

1. **Implementare**

**Clasa Scheduler – are un vector de Servere. Metodele principale sunt chanheStrategy și dispatchClient. Metoda de schimbare a strategiei stabilește în funcție de field-ul strategy care metodă de optimizare va fi aleasă în cadrul metodei dispatchClient pentru a ști care metoda addClient va fi apelata: fie cea din clasa ConcreteSTrategyQueue, fie cea din clasa ConcreteSTrategyTime.**

**Clasa Serever – fiecare obiect de tip server va fi un thread, astfel această clasă extinde clasa Thread și implementează interfața Runnabale pentru a putea suprascrie meoda run(). Fiecare obiect de acest tip prezintă un indice pentru a putea fi identificată coada. Metoda principală este metoda run care ia fiecare client din coaddă la rând, îl păstrează în varful cozii un timp egal cu timpul de procesare al clientului apoi continua până când coada este goală.**

**Clasa SimulationManager – această clasă are ca field-uri toate datele necesare simulării care vor fi citite din fișierul text de intrare. Alte doua field-uri important ear fi: selectionPolcy și o listă cu clienții generați random, sortați în ordine crescătoare în funcție de timpul la care au ajuns la coadă. Constructorul acestei clasei pornește serverele(threadurile corespunzătoare acestora). Și acestă clasa extinde clasa Thread și implementează interfața Runnable. Printre metodele esențiale ale acestei clase se numără și metoda de generare random a clienților supuși simulării, dar și metoda run care este responsabila de distribuirea clienților în cozi.**

**Interfața Strategy - metoda addClient care primește ca și parametrii o listă de Servere și un client care urmează a fi introdus în una dintre cozi.**

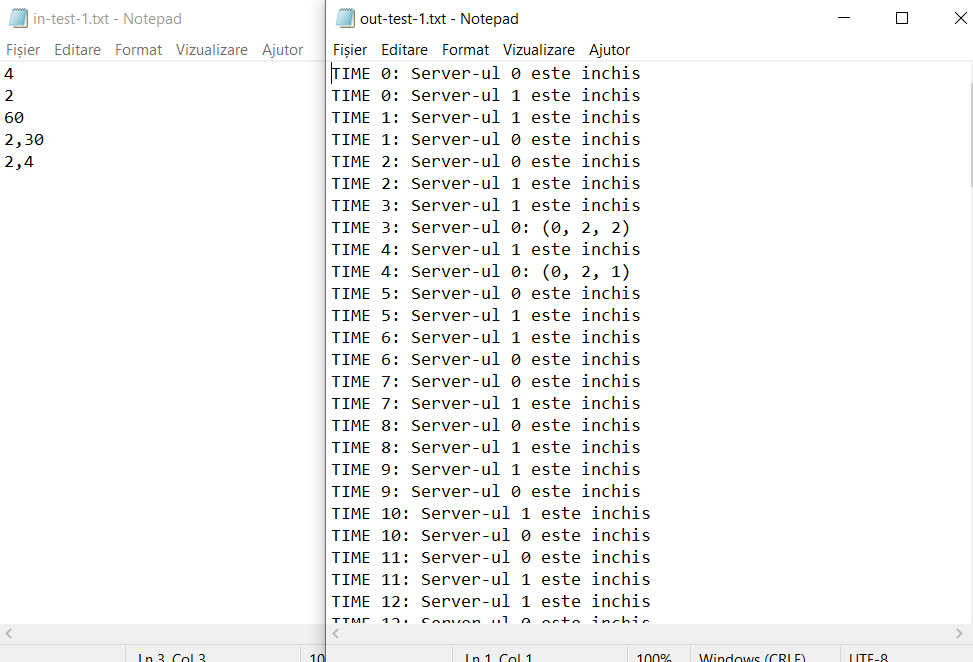
**Clasa ConcreSTrategyQueue – implementează interfața Strategy. Nu este folosită suprascrierea metodei addClient() din această clasă. Împreună cu interfața Strategy și clasele ConcreteSTrategyTime și Scheduler formeaza un Strategy Pattern.**

**Clasa ConcreSTrategyTime – implementează interfața Strategy. Aceasta este metoda folosită în cadrul metodei care se ocupă cu distribuirea clienților în cozi. Împreună cu interfața Strategy și clasele ConcreteSTrategyTime și Scheduler formeaza un Strategy Pattern. (strategia aleasă în încercarea optimizării timpului mediu de așteptare în cadrul unei cozi de către un client).**

**Clasa SelectionPolicy – o clasă de tip enumerație care descrie cele două strategii posibile de adăugare a unui client într-o coada: SHORTEST\_QUEUE, SHORTEST\_TIME.**

**Clasa Client – descrie obiectele cu care se modelează această simulare, clienții. Aceștiaa sunt reprezentați printr-un ID, timpul de sosire la coadă și timpun necesar de procesare atunci când se află în vârful cozii.**

1. **Rezultate**

****

**Fișier text cu date de intare**

**Fișier text cu date de ieșire**

# Concluzii

**În concluzie realizarea acestui proiect a fost foarte utilă , amintind de o mare parte din principalele concepte ale porgramării orientate pe obiecte învățate semestrul trecut. Datorită modului în care a fost gândită și proiectată aplicația, dezvoltările ulterioare pot fi foarte ușor implementate în orice moment și nu doar de către cel ce a facut-o, ci și de către alți programatori care ar dori să aducă îmbunătățiri ulterioare**

**aplicației.**

**O dezvoltare ulterioară care ar putea fi făcută pentru ca aplicația să fie mai ușor de utilizat ar putea fi, spre exemeplu, o interfață grafică care să permit utilizatorului o interacțiune mai mare cu aplicația. În plus, acesta ar fi mult mai practică și mai utilă, dacă avem în vedere un utilizator neexperimentat. În ceea ce privește datele de intrare acestea ar fi mult mai ușor de introdus prin intermediul unei interfațe decât prin modul prin care se realizează în acest moment: citirea dintr-un fișier text de intrare. Totodată și codul ar putea fi îmbunătățit prin scrierea sa mai eficientă, oferind aplicației o mai mare viteză și un timp de execuție mult mai bun în cazul datelor de intrare mult mai mari.**

# Bibliografie

* [**https://ro.wikipedia.org/wiki/Fir\_de\_execu%C8%9Bie**](https://ro.wikipedia.org/wiki/Fir_de_execu%C8%9Bie)
* [**https://www.baeldung.com/java-copy-on-write-arraylist**](https://www.baeldung.com/java-copy-on-write-arraylist)
* [**https://en.wikipedia.org/wiki/Strategy\_pattern**](https://en.wikipedia.org/wiki/Strategy_pattern)
* [**http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/Assignment\_2/Java\_Concurrency.pdf**](http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf)
* [**https://www.w3schools.com/java/java\_files\_read.asp**](https://www.w3schools.com/java/java_files_read.asp)
* **Cursurile și laboratoarele de “Programare orientate pe obiect” și “Tehnici de Programare”**