DOCUMENTATIE

TEMA 2

Sistem de management al cozilor ce

implementează mecanisme de alocări eficiente

NUME STUDENT: Fazakas Edina

GRUPA: 30223

# CUPRINS

[1. Obiectivul temei 3](file:///E:\MATERIALE%20AN%202%20SEM2\TP\PT2022_30223_Fazakas_Edina_Assignment_1\PT2021-2022_Documentation.doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///E:\MATERIALE%20AN%202%20SEM2\TP\PT2022_30223_Fazakas_Edina_Assignment_1\PT2021-2022_Documentation.doc#_Toc95297886)

[3. Proiectare 3](file:///E:\MATERIALE%20AN%202%20SEM2\TP\PT2022_30223_Fazakas_Edina_Assignment_1\PT2021-2022_Documentation.doc#_Toc95297887)

[4. Implementare 3](file:///E:\MATERIALE%20AN%202%20SEM2\TP\PT2022_30223_Fazakas_Edina_Assignment_1\PT2021-2022_Documentation.doc#_Toc95297888)

[5. Rezultate 3](file:///E:\MATERIALE%20AN%202%20SEM2\TP\PT2022_30223_Fazakas_Edina_Assignment_1\PT2021-2022_Documentation.doc#_Toc95297889)

[6. Concluzii 3](file:///E:\MATERIALE%20AN%202%20SEM2\TP\PT2022_30223_Fazakas_Edina_Assignment_1\PT2021-2022_Documentation.doc#_Toc95297890)

[7. Bibliografie 3](file:///E:\MATERIALE%20AN%202%20SEM2\TP\PT2022_30223_Fazakas_Edina_Assignment_1\PT2021-2022_Documentation.doc#_Toc95297891)

# Obiectivul temei

Tema aceasta constituie implementarea și proiectarea a unei aplicații care are ca scop analiza sistemelor bazate pe cozi de așteptare prin simulând a serie de N clienți care sosesc pentru service, care intră în cozile Q, așteaptă, sunt serviți și în cele din urmă părăsesc cozile și calcularea timpului mediu de așteptare, a timpului mediu de serviciu și a orelor de vârf.

1. Analiza problemei și identificarea cerinței. – Capitolul 2
2. Proiectarea unui design pentru calculator – Capitolul 3
3. Implementarea calculatorului polinomial – Capitolul 4
4. Testarea calculatorului polinomial – Capitolul 5

# Analiza problemei, modelare, scenarii, cazuri de utilizare

*A screenshot of a game

Description automatically generated with low confidence*

*Graphical user interface, application

Description automatically generatedGraphical user interface

Description automatically generated*

*Diagram

Description automatically generated*

Utilizare caz : simulare de configurare

Actor primar : utilizator

Principalul scenariu de succes

1.Utilizatorul introduce valorile pentru: numărul de clienți, număr de cozi, interval de simulare, minim și

ora maximă de sosire și minim și maxim timp de serviciu minim și maxim

2. Utilizatorul face clic pe butonul de validare a datelor de intrare

3. Aplicația validează datele și afișează un mesaj informând utilizatorul să înceapă simularea

Secvență alternativă:

Valori nevalide pentru parametrii de configurare

-Utilizatorul introduce valori nevalide pentru configurarea aplicației parametrii

- Aplicația afișează un mesaj de eroare și solicită utilizator pentru a introduce valori valide

-Scenariul revine la pasul 1

Problema este percepută de Utilizatorul introduce valori nevalide pentru configurarea aplicației parametrii

-Aplicația afișează un mesaj de eroare și solicită utilizator pentru a introduce valori valide

# Proiectare

Proiectarea - Arhitectură conceptuală

*Diagram

Description automatically generated*

**

**

Am utilizat modelul Model View Controller care se împarte aplicarea în trei domenii: procesare, ieșire și intrare. Componentele modelului încapsulează datele de bază și funcţionalitate. View: afișează informații despre componente pentru utilizator, obține datele pe care le afișează din model. Controler: fiecare vizualizare are asociat o componentă controler. Controlerele primesc intrarea, de obicei ca evenimente care denotă mișcarea mouse-ului, activarea butoanele mouse-ului sau intrarea de la tastatură. Evenimentele sunt traduse în cereri de service, care sunt trimise fie la model sau la vedere.În proiect avem 3 package-uri. Package-urile sunt : Model, View, Controller.

În fiecare package sunt clase. Clasa Task are trei variabile, idul de tip int, arrivalTime de tip int, și variabila serviceTime de tip int. Clasa Server are ca variabilă un BlockingQueue de tip Task tasks și o variabilă de tip AtomicInteger waitingPeriod. Folosesc 3 pachete pentru a avea organizare de Model View Controller. În pachetul Model am două clase, adică Task și Server. În pachetul Business Logic am clasa SimulationManager, Scheduler, interfața Strategy care are două clase ce le implementează TimeStrstegy și ShortestQueueStrategy. Utilizatorul va schimba numărul de clienți, numărul de cozi, numărul minim de arrival time, numărul maxim de arrival time, numărul minim de service time, numărul maxim de service time. Programul va implementa două strategii de mecanisme pentru a organiza clienții în cozile cu servicii. Clienții practic sunt taskurile, serviciile practic cozile în sine. Clasa SimulationManager va organiza toate evenimentele, punerea taskurilor in cozi, respectiv toate operațiile cu ajutorul threadurilor. Sincronizarea ajută la evitarea problemelor cauzate de utilizarea threadurilor. Clasa Scheduler va analiza după care strategie vom organiza cozile cu clienții. Se va alege metoda dată, și astfel se vor pune taskurile în coada tasks BlockingQueue. Clasa Generate va genera un număr de clienți cu arrival time random, cu service time random, cu un id anume, și va sorta taskurile după ordinea crescătoare a arrival time-ului.

(Se va prezenta proiectarea OOP a aplicatiei, diagramele UML de clase si de pachete, st ructurile de date folosite, interfetele definite si algoritmii folositi (daca e cazul)

# Implementare

Implementarea este făcută în 2 package-uri: Model și Business Logic. În package-ul Model sunt 2 clase: Server și Task. În package-ul View se află interfața grafică. În package-ul Business Logic se află clasele Generate, Scheduler, SelectionPolicy, SimulationManager, interfața Strategy, și două clase care le implementează: Concrete Strategy Queue și Concrete Strategy Time. Clasa Task are 5 variabile, id-ul de tip int, arrival time de tip int, variabila service time de tip int, variabila service de tip int și variabila ok de tip int. Această clasă are getteri, și metoda compareTo pentru a sorta taskurile după ordinea crescătoare a arrival time ului. Clasa Sever are ca variabilă un BlockingQueue de tip Task tasks, respectiv o variabilă waitingPeriod de tip AtomicInteger, și o variabilă i pentru ca să aibă fiecare server un identificator numeric. În această clasă se inițiază cu constructorul variabilele instanță. Totodată, această clasă implementează Runnable ceea ce înseamnă că va fi un thread. Metoda creată impicit run va face afișările de informații despre servere și taskurile sale ca id, arrival time, service time. Se vor afișa aceste informații și în fișier, dar și în consolă. Aici se va implementa totodată o metodă compareTo pentru a sorta serverele după waiting period-ul lor. Mutându-ne în package-ul Business Logic observăm clasa Generate. În clasa Generate practic se va face randomizarea taskurilor. Avem ca variabilă processing time de tip int, arrival time de tip int, N adică numărul de taskuri, de clienți, un Array List de tip Task task, o variabilă id de tip int, și o variabilă service time de tip int. Generarea random de taskuri se face în constructor, acest lucru se face după numărul de clienți. Și astfel se pun taskurile generate random în lista de taskuri, pe care o vom sorta după ordinea crescătoare a arrival time -ului. Clasa Scheduler practic va fi cum redă și numele schedulerul, adică organizatorul. Clasa conține o variabilă de tip lista de tip Server servers, o variabilă max nr servers de tip int, o variabilă de tip Strategy strategy. Constructorul clasei va inițializa numărul de serveri și lista de serveri conform metodei sincronizării ca Collections synchronizedList de tip Array List Server. Aici se vor crea serverii de număr max number servers și se va adăuga serverii în lista servers. În această clasă se va afla metoda sincronă change Strategy care va selecta strategia conform parametrului policy de tip Selection Policy. Cele două policy sunt SHORTEST\_TIME și SHORTEST\_QUEUE. Se va crea o nouă variabilă clasă de tip Concrete Strategy Queue sau Concrete Strategy Time. În metoda dispatch Task se va adăuga taskul dat ca parametru în instanța strategy cu apelul metodei addTask. Selection Policy este o enumerațte constând din variabilele shortest queue și shortest time. Tot în acest package se află interfața Strategy care are ca metodă add task. Clasele Concrete Strategy Queue și Concrete Strategy Time implementează interfața Strategy, așadar trebuie să implementeze metoda add task. Aceste două strategii constă în organizarea taskurilor în cozi. Prima strategie, concrete strategy queue practic va parcurge lista de serveri și adaugă taskul respectiv în prima coadă care este goală. În cazul în care toate cozile sunt pline, adică au un task anume – deoarece o coadă poate avea numai unn task într-un timp anume- trebuie să incrementăm arrival time-ul pentru a fi luat în considerare. A doua strategie, concrete strategy time va sorta serverii după waiting period și așa le va selecta, prima care e goală. Ca și în cealaltă strategie, și aici trebuie să analizăm cazul în care toate cozile sunt pline, în acest caz trebuie să incrementăm arrival time ul pentru a considera acel task mai târziu. Clasa SimulationManager care se află în packageul Business Logic practic va fi producătorul. Aici vom începe threadul. În clasă sunt ca variabile time limit de tip int, care va arăta numărul de secunde care sunt date pentru a organiza cozile. Sunt și variabilele max processing time de tip int, min processing time de tip int, min arrival time de tip int, max arrival time de tip int, number of servers de tip int, number of servers de tip int, selection policy de tip Selection Policy, lista generated task de tip Task, și scheduler de tip Scheduler. În constructor vom inițializa clasa pentru simulare. Vom crea un obiect scheduler care va avea un număr de serveri, vom alege strategia după care vom organiza cozile și vom chema metoda change strategy pentru strategia respectivă din clasa Scheduler. Vom face generarea random a taskurilor prin apelarea constructorului din clasa Generator. Aici vom face afișarea în fișier, respectiv în consolă a taskurilor generate random. În metoda run generată va începe implementarea organizării. Începem contorizarea current timelului care va merge până ajunge la time limit. Vom parcurge lista de taskuri generated Task și dacă arrival timeul taskului va fi egală cu current time și taskul respectiv nu a fost deja analizată, atunci vom adăuga taskul respectiv în metoda dispatch task din clasa scheduler, vom marca taskul ca fiind deja pus în coadă. După fiecare analiză trebuie să incrementăm timpul recent. După acest moment parcurgem serverele și pornim threadul. Dacă serverul respectiv are un task, trebuie să decrementăm timpul de serviciu petrecut în coadă. În cazul în care deja service timeul este deja 0, trebuie să scoatem taskul, clientul din server. După fiecare moment trebuie să punem threadul la sleep pentru o secundă ca să corespundă cu convenția noastră.

Clasa GUI implementează clasa ActionListener pentru a putea folosi mouse-ul și alte butoane de pe tastatură. Interfața are un frame, 3 textfielduri, unul cu primul polinom, cu al doilea polinom și cu polinomul rezultat. Operațiile și numerele pentru a scrie un polinom se afișeaza și se efectuează prin JButton-uri. Avem și butoane de delete și clear pentru a șterge ce am scris în TextFielduri. Prin metoda setBounds se aranjează fiecare buton cu coordinate, lungime și lățime. Toate butoanele și labelurile se adaugă în frame și fiecare buton se adaugă prin metoda ActionListener la operația de click. Pentru a diferenția în care TextField vrem să scriem o să folosim MouseListener, adică ne uităm pe care TextField am dat click și în așa fel o să modificăm valoarea unei variabile și o să testăm unde printăm polinomul respectiv. Metoda actionPerformed are ca parametru o variabilă de tip ActionEvent prin care o să judecăm evenimentele. În primul rând luăm în considerare butoanele pentru ”tastatură”. Numerele și caracterul x, respectiv caracterele speciale ca și + - ^ / etc sunt luate în considerare prin click și atunci când apăsăm butonul respectiv se printează caracterele în care TextField dăm click. Când apăsăm butonul de clear sau delete se vor șterge ori textFieldurile ori caracter cu caracter. Unele butoane sunt pentru a alege o operație. În această situație se va converti stringul de polinom, se va efectua metoda respectivă și se afișează rezultatul. Pentru a converti stringuri în monoame folosim regular expression. Pentru asta folosim un pattern care ne va ajuta să găsim expresii în string-uri care arată ca pattern-ul. Setul de match ne va ajuta să găsim aceste expresii. Expresia din algoritmul meu este dat de un semn, apoi un număr – ori dintr-un digit, ori mai multe *cifre.* Apoi va veni opțional x -ul, opțional caracterul ”^” apoi iarăsi cifre. Astfel, funcția match *după tipul matcher va arăta dacă va găsi un grup care arată ca expresia dată. Aceste expresii se vor converti în monoame. După ce convertim expresia în monom, pentru a afișa rezultatul, trebuie să o convertim iarăși în string.*

*(Se va descrie fiecare clasa cu campuri si metodele importante. Se va descrie implemantarea interfetei utilizator.)*

# Rezultate

Un test este o bucată de cod a cărei funcție este de a verifica dacă o altă bucată de cod funcționează corect. Pentru a face verificarea, apelează metoda testată și compară rezultatul cu rezultatul predefinit așteptat. Un rezultat așteptat poate fi, de exemplu, o anumită valoare returnată sau o excepție. Pentru a lucra cu JUnit trebuie să configurăm Maven-ul. Creăm clasa de testare. Trebuie să implementăm metoda pentru testare și adăugăm în clasă. Apoi trebuie să dăm run la test. Aserțiunile sunt metod statice definite în clasa org.junit.jupiter.api.Assertions: assertEquals , assertAll , assertNotEquals , assertTrue. În cazul în care facilitățile de afirmare oferite de JUnit Jupiter nu sunt suficiente, terț pot fi folosite biblioteci (de exemplu, AssertJ, Hamcrest. Suitele de testare reunesc mai multe clase de testare într-o suită, astfel încât acestea să pot fi rulate. Testele parametrizate fac posibilă rularea unui test de mai multe ori cu diferite argumente. Trebuie să declare cel puțin o sursă care va furniza argumentele pentru fiecare invocare și apoi consumă argumentele din metoda de testare.

*Se vor prezenta scenariile pentru testare cu Junit sau alt framework de testare.*

# Concluzii

Din această team se poate învăța multe lucruri. În primul rând operațiile cu threaduri, sincronizarea threadurilor, multithreading. Strategiile pentru a organiza clienții în cozi, folosirea acestor strategii, folosirea enumerațiilor. Folosirea tipului de variabilă BlockingQueue, AtomicInteger. Modul de a folosi un thread, de exemplu cu metoda run ce este implementată de interfața Runnable. Sincronizarea este o altă parte pe care am putut să o înțelegem. Acesta se face prin variabile finale, metode sincronizate, prin Lock și atomicitatea, respectiv tipul de Collections sincronizat. Pentru a folosi interfață și a modifica datele după thread putem folosi Swing Worker. Designul Model View Controller și cum se creează este un alt lucru pe care a trebuit să îl cercetez în cursul acestei proiect. Junit este un alt domeniu pe care l-am putut folosi pentru beneficiul nostrum. Testările sunt folositoar pentru a testa dacă am lucrat correct și dacă funcționează metodele folosite correct. Aceste lucruri au fost învățate pe lângă lucrurile deja previzuite în Java.

(*Se vor prezenta concluziile, ce s-a invatat din tema, dezvoltari ulterioare.)*

# Bibliografie

<https://google.github.io/styleguide/javaguide.html>

<https://www.google.com/search?q=use+case+diagram&rlz=1C1BNSD_enRO980RO980&sxsrf=APq-WBsxOC_EZsCXCum1TfekwOVbddYYdg:1646594624210&source=lnms&tbm=isch&sa=X&ved=2ahUKEwj2t6j9mrL2AhUVAhAIHdRMDrUQ_AUoAXoECAIQAw&biw=767&bih=704&dpr=1.25#imgrc=dfmnzqrEOzBCwM>

<https://stackoverflow.com/questions/8942751/use-intellij-to-generate-class-diagram>

<https://www.google.com/search?q=uml&rlz=1C1BNSD_enRO980RO980&sxsrf=APq-WBugGiQYi7p4XJ_GCMleyWCf4e56iA:1646581013150&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjPvIej6LH2AhUqMuwKHQGUDKwQ_AUoAXoECAIQAw&biw=767&bih=704&dpr=1.25#imgrc=PHR4AloTe-FkhM>

<https://stackoverflow.com/questions/18754490/using-compareto-and-collections-sort>

baeldung.com/regular-expressions-java

<https://www.vogella.com/tutorials/JavaRegularExpressions/article.html>

<https://www.tutorialspoint.com/java/java_regular_expressions.htm>

<https://stackoverflow.com/questions/17969436/java-regex-capturing-groups>

<https://math.stackexchange.com/questions/215734/pseudo-code-for-polynomial-long-division>

<https://www.jetbrains.com/help/idea/testing.html>

<https://www.guru99.com/java-swing-gui.html>

<https://docs.oracle.com/javase/tutorial/uiswing/>