DOCUMENTATIE

TEMA *2*

NUME STUDENT : Calin Catalin Iacob
GRUPA : 30222

CUPRINS

1. Obiectivul temei

Principalu obiectiv al temei este de a proiecta si a implementa o aplicatie care are ca scop analiza sistemelor bazate pe cozi de asteptare , simuland o serie de N client care sosesc pentru servici, care intra in cozile Q, asteapta, sunt serviti si in cele din urma parasesc cozile de asteptare, si calculand timpul mediu de asteptare. Aceasta aaplicatie propune rezolvarea unei situati destul de intalnita in ziua de azi la diverse chioscuri, anume presupune creearea unei aplicati de simulare, care sa genereze random un anumit numar de clienti, care sosec la un anumit timp, asteapta si in final dup ace sunt serviti pleaca. Iar simularea se va opri in momentul in care timpul de simulare se termina. Un alt obiectiv este cel in care trebuie sa afisam in timp real evolutia serverelor, prezentand utilizatorului aplicatiei, cum decurge simularea.

Obiective secundare:

- *Analizarea problemei si indentificarea cerintelor - capitolul 2;*
- *Proiectarea aplicatiei de simulare – capitolul 3;*
- *Implementarea aplicatiei de simulare – capitolul 4;*
- *Testarea simulari – capitolul 5;*

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

- Aceasta aplicatie presupune lucrul cu threadul, unul destul de complicat din cate se vede, deoarece fiecare thread se executa doar in momentul in care ajung clienti la coada, astfel, o problema in proiectarea aplicatiei de simulare este felul in care distribuim si pornim threadurile pe servere(cozi), in mod concurrent, iar solutia pentru pe care am abordat-o este aceea, de avea o metoda care sa porneasca un thread pe un server doar daca pe aceea instanta "Server", nu ruleaza deja un thread, acest lucru l-am realizat adungand inca un parametru Serverului, "flag", care in momentul in care acesta are valoarea "true", imi indica faptul ca pe serverul respective ruleaza deja un thread si nu este nevoie sa pornim altul, iar cand valoarea flga-ului este "false, imi indica faptul ca pe serverul respectiv, nu ruleaza nici un thread, iar in cazul acesta se porneste un thread pe serverul respective care se va ocupa de managementul clientilor sai. Metoda de pornire a threadurilor secundare, putem spune, este apelata din threadul principal.

Cerinte functionale:

- Aplicatia de simulare trebuie sa permita utilizatorului sa seteze datele pentru simulare;
- Aplicatia de simulare trebuie sa permita utilizatorului sa porneasca simularea;
- Aplicatia de simulare trebuie sa permita utilizatorolui sa urmareasca evolutia fiecarei cozi in timp real;

Use-case: setarea datelor pentru simulare;
Actor principal: utilizatorul;
Scenariul principal: scenariu in care utilizatorul intoduce date corecte;

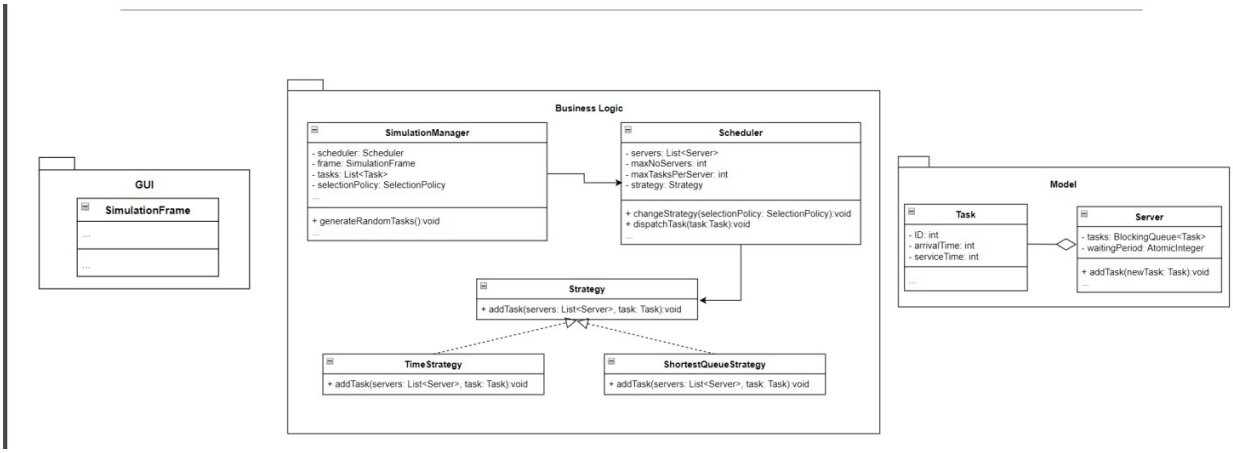
1. *Utilizatorul introduce valorile pentru: timpul de executie al simulari, numarul de cozi, numarul de clienti care vor fi asignati cozilor, timpul minim de sosire, timpul maxim de soisire, timpul minimi pe care clientul in petrece in coada si timpul maxim pe care clientul in petrece in coada;*
2. *Utilzatorul apasa pe butonul start;*
3. *Aplicatia valideaza datele si porneste simularea;*

Scenariu alternativ: scenariu in care utilizatorul introduce date incorecte pentru parametri de intrare;

1. *Utilizatorul introduce date invalide;*
2. *Utilizatorul apasa pe butonul start;*
3. *Se afiseaza un mesajul de erroare " Invalid data ", simularea nu porneste;*

3. Proiectare

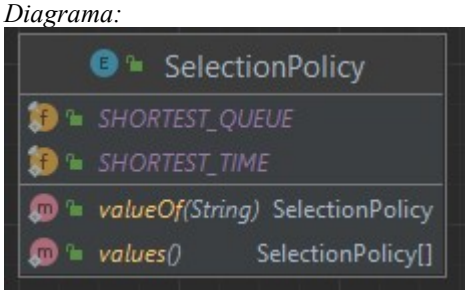
Diagrama de clase impartite pe pachete:



3.1 Clasa SelectionPolicy

- aceasta clasa reprezinta un enum, care este folosit pentru a alege strategia de distibuire a clientilor pe cozi.

Diagrama:



3.2 Clasa Task

- clasa care memoreaza informatii despre un client, informatii precum id-ul, timpul de sosire si timpul de procesare;

Diagrama:

3,3 Clasa Server

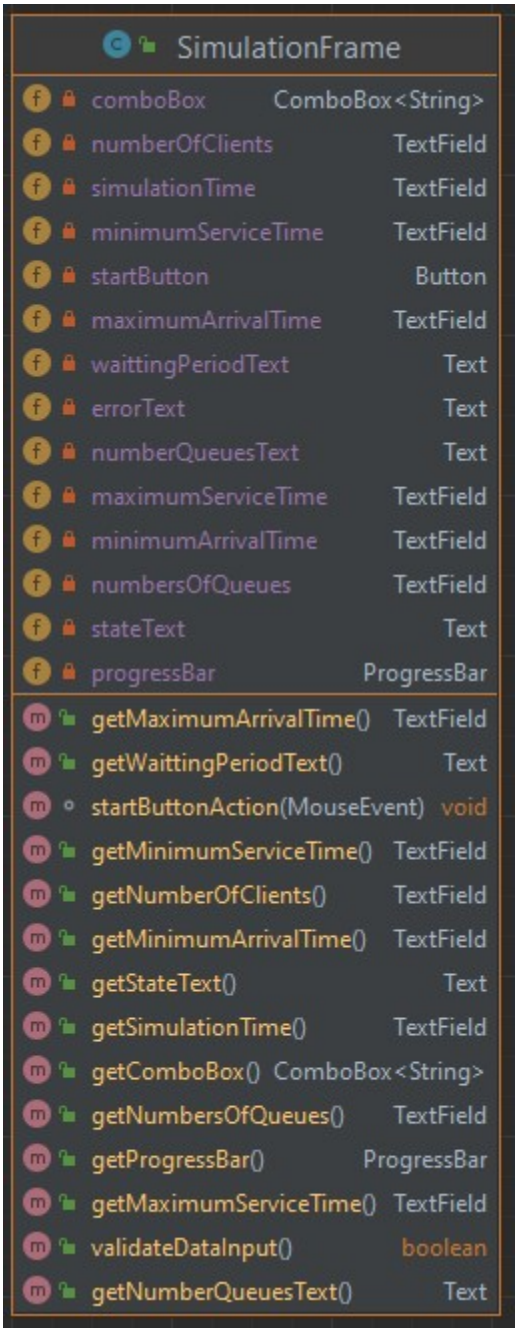
- *Clasa a carei instante vor reprezrnta cozile, care vor fi prelucrate in paralel de threadurile asignate respectivelor instante;*

Diagrama:

3.4 Clasa SimulationFrame

- *Reprezinta controlerul interfetei, carea preia comenzile si datele introduse de utilizator pentru simulare;*

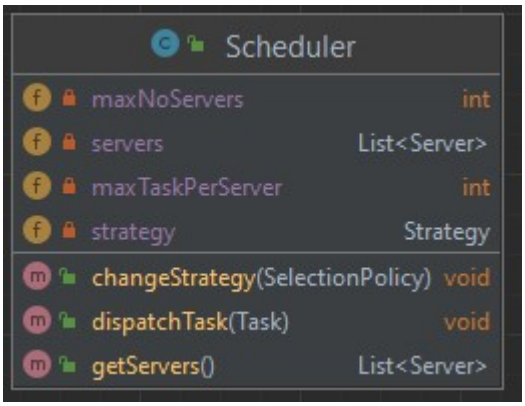
Diagrama:



3.5 Clasa Scheduler

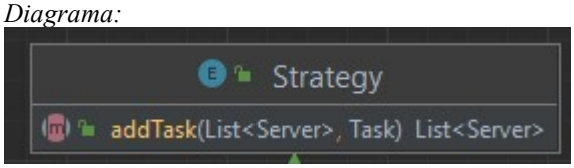
- aceasta clasa se ocupa cu distribuirea clientilor pe cozi, astfel incat acestia sa petreaaca cat mai putin timp asteptand la coada;

Diagrama:



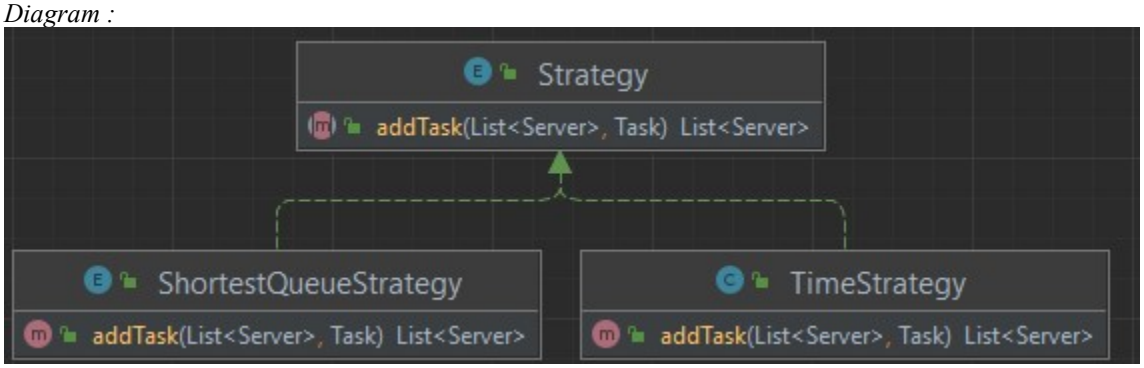
3.6 Interfata Strategy
- interfata care defineste metoda de adugare in coada;

Diagrama:



3.7 Clasele ShortestQueueStrategy si TimeStrategy
- clase care definesc strategile de distibuire clientilor la momentul sosiri lor , pe cozi. ShortestQueueStrategy alege coada in care se afla cei mai putini clienti, iar TimeStrategy alege coada cu cel mai mic timp de asteptare;

Diagram :



3.8 Clasa SimulationManager

- clasa care se ocupa de pornirea threadurile pe obiectele de tip server si generare de taskuri (clienti) random si tot odata clasa in care se updateaza interfata frafica cu informatile despre simulare, si scrierea in fisierul logfile.txt a evenimentelor;

Diagrama:

4. Implementare

4.1 Clasa Task : defineste un client cu timpul de procesare si sosire al acestuia.

- *Atribute:*
  - *ID, tip int: retine un numar de identificare unic;*
  - *serviceTime: tip int, retine timpul de procesare al clientului;*
  - *arrivalTime: tip int, retine timpul la care va sosi clientul;*
  - *subclasa SortByArrivalTime: tip statica, clasa care ajuta la sortarea unei liste de taskuri;*
- *Metode: constructor, getter, setter;*

4.2 Clasa Server: reprezinta o coada care va fi asociata unui thread.

- *Atribute:*
  - *tasks: tip BlockingQueue<Task>: coada care retine clienti care is asteapta randul pentru a fi serviti;*
  - *waitingPeriod: tip AtomicInteger: retine timpul total de asteptare in coada pana la finalizare;*
  - *maxTaskNumber: tip int: reprezinta numarul maxim de clienti care pot sta la o coada;*
  - *flag: tip boolean: acesta este un indicator care imi arata daca ruleaza un thread sau nu pe o instanta a clasei;*
- *Metode:*
  - *constructor, getter, setter;*
  - *run(): aceasta metoda este implemetata din interfata Runnable si reprezinta comportamentul threadului;*

4.3 Clasa SelectionPolicy: este un enum care defineste cele doua politici de management al cozilor.

- *Atribute:*

- *SHORTEST_QUEUE, petru strategia de distribuire in cozi dupa cea mai scurta;*
- *SHORTEST_TIME: pentru strategia de distribuire in cozi dupa cel mai scurt timp de asteptare;*

- *Metode: - ;*

4.3 Interfata Strategy: interfata care defineste metoda de adugare unui server de clientii noi.

- *Atribute: -*
- *Metode:*
  - *add(List<Server>, Task t), metoda care aduga in coada unui server un client nou;*

4.4 Clasa TimeStrategy: clasa care defineste metoda de adugare bazata pe cel mai mic timp de asteptare.

- *Atribute: -*
- *Metode:*
  - *add(List<Server>, Task t), metoda care aduga in coada unui server un client nou , dupa politica coada cu cel mai mic timp de asteptare;*

4.6 Clasa ShortestQueueStrategy: clasa care defineste metoda de adugare bazata pe cea mai scurta coada.

- *Atribute: -*
- *Metode:*
  - *add(List<Server>, Task t), metoda care aduga in coada unui server un client nou , dupa politica coada cu cel mai mic numar de clienti care asteapta la coada respectiva;*

4.7 Clasa Scheduler: clasa care programeaza stabileste ordinea si plasarea in cozi a clientilor.

- *Atribute:*
  - *maxNoServers: tip int, retine numarul maxim de servere care pot fi create pentru managementul clientilor;*
  - *servers: tip List<Server, reprezinta serverele care se vor ocupa de managementul clientilor;*
  - *maxTaskPerServer: tip int, retin numarul maxim de clienti care pot astepta la o coada, sau numarul maxim pe care un server ii poate servi;*
  - *strategy : tip Strategy: reprezinta strategia aleasa pentru distribuirea clientilor pe servere;*
- *Metode:*
  - *Constructor, getter, setter;*
  - *changeStrategy(SelectionPolicy selctionPolicy), metoda care schimba strategia de distribuire a clientilor pe servere in functie de politica de selectare data prin parametrulselectionPolicy;*
  - *dispatchTask(Task t), metoda care apeleaza metoda adugare definita prin parametrul strategy;*

4.8 Clasa SimulationManager: clasa care se ocupa de intreg managementul serverelor, asociand acestora threaduri.

- *Atribute:*
  - *averageTime: tip float: retine timpul mediu de asteptare in coada a unui client;*
  - *scheduler: tip Scheduler: parametru care se ocupa de managementul clientilor pe servere;*
  - *frame: tip SimulationFrame: parametru petru updatarea interfetei grafice;*
  - *selectionPolicy: tip SelectionPolicy: retine politica de distribuire a clientilor pe cozi;*

- *tasks: tip List<Task>, retine o lista de clienti generate random de catre metoda generateRandomTasks();*
- *logFile: tip FileWriter: folosit pentru scrierea in fisierul logFile.txt a evenimentelor generate de simulator;*
- *simulationTime: tip int: retine timpul in secunde, timpul de rulare al simulari;*

- *Metode:*
  - *Constructor;*
  - *updateFrame(int) : aceata metoda updateaza interfata grafica, pentru a vedea evolutia serverelor in timp real;*
  - *generateRandomTasks(): metoda care generaza clienti random, care urmeaza sa fie distribuiti pe servere;*
  - *startThreads(): metoda care porneste threadurile pentru serverele care au cel putin un client;*
  - *printInFile(): metoda care se ocupa de scrierea in fisierul logFile.txt a evenimentelor generate de simulator;*
  - *run(): metoda implementata din interfata Runnable, care corespunde threadului principal;*

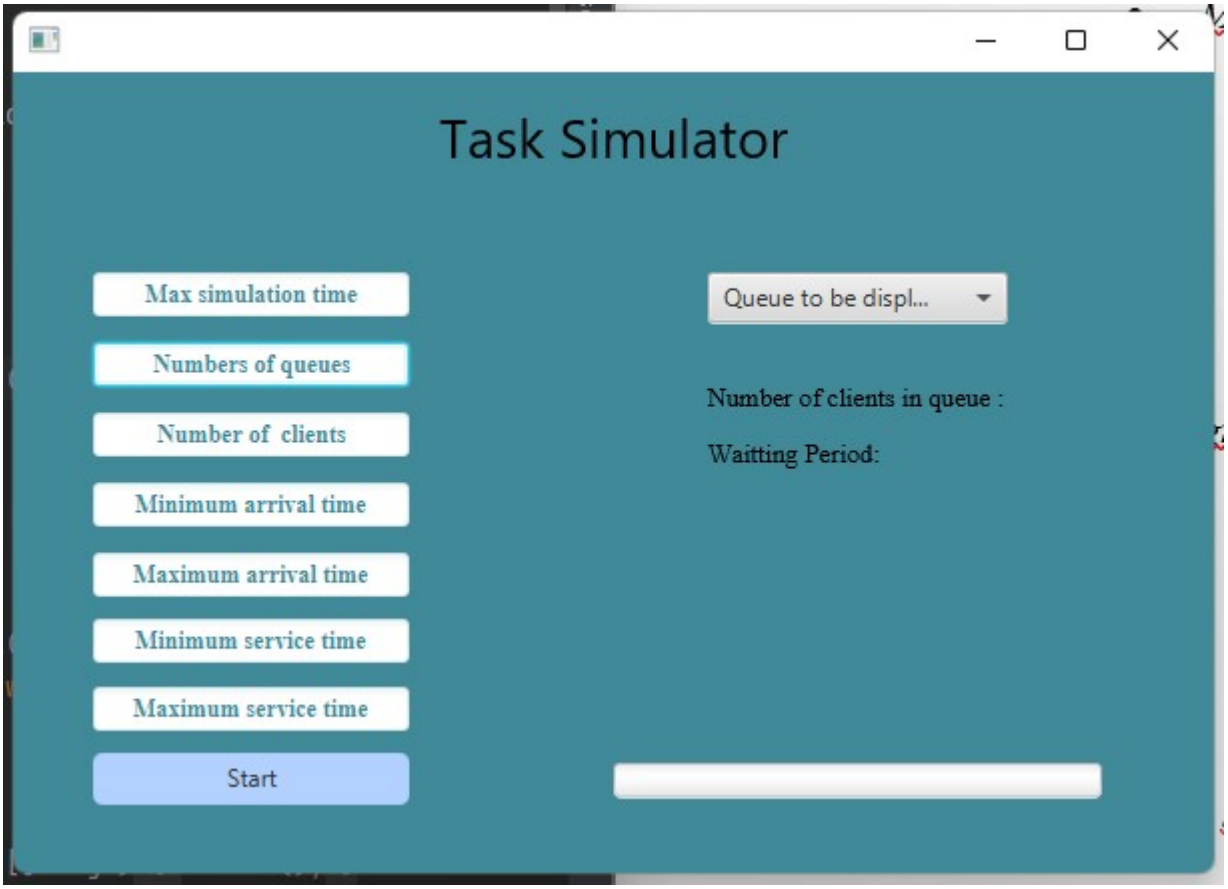
4.9 Clasa SimulationFrame: clasa de control a interfatei grafice.

- *Atribute:*
  - *comboBox: tip ComboBox< String >: atribut utilizat pentru a selecta pentru care dintre servere sa se prezinte evolutia sa;*
  - *numberOfClients: tip TextField: atribut utilizat pentru a introduce de catre utilizator numarul de clienti care sa se genereze;*
  - *simulationTime: tip TextField: atribut utilizat pentru a introduce de catre utilizator timpul de simulare;*
  - *minimumServiceTime: tip TextField: atribut utilizat pentru a introduce de catre utilizator timpul minim de procesare a unui client;*
  - *maximumServiceTime: tip TextField: atribut utilizat pentru a introduce de catre utilizator timpul maxim de procesare a unui client;*
  - *minimumArrivalTime: tip TextField: atribut utilizat pentru a introduce de catre utilizator timpul minim de sosire a unui client pentru a fi introdus in coada;*
    maximumArrivalTime: tip TextField: atribut utilizat pentru a introduce de catre utilizator timpul maxim la care poate sosi un client pentru a fi introdus in coada;
  - *numberOfQueues: tip TextField: atribut utilizat pentru a introduce de catre utilizator numarul de cozi cu care va lucra simulatorul;*
  - *errorText: tip Text; atribut folosit pentru a semnal utilizatorul daca a introdus date invalide;*
  - *numberQueuesText: tip Text: atribut folosit pentru a arata utilizatorului numarul de clienti dintr-o coada;*
  - *waitingPeriodText: tip Text: atribut folosit pentru a arata evolutia timpului de asteptare dintr_o coada;*
  - *stateText: tip Text; atribut folosit pentru a arata utilizatorului starea aplicatie de simulare, "Running…", cand acesta ruleaza, si "Finished!" cand a terminat rularea;*
  - *progressBar: tip ProgressBar: atribut folosit pentru a arata intr-o bara de progres evolutia timpului de simulare;*
  - *startButton: tip Button: atribut care repreinta butonul de validare si pornire a simulari;*
  -
- *Metode:*
  - *Getter;*

- *validateDataInput(): metoda care valideaza datele introduse de catre utilizator si returneaza "true" daca datele sunt valide si " false" in caz contrar;*
- *startButtonAction(MouseEvent): metoda legata de butonul de start care in momentul clickului pe butonul start se apeleaza si porneste simularea, dup ace valideaza datele introduse de catre utilizator;*

4.10 GUI

- pentru construirea interfetei grafice am folosit JavaFX SceneBuilder si CSS pentru anumite efecte suplimentare asupra componentelor interfetei;



5. Rezultate

Scenarii de utilizare:

Scenariu 1:

- *Se apas butul de run, porneste aplicatia de simulare;*
- *Apare GUI-ul;*
- *Introducem datele cu mare atentie ,ca sa nu introducem ceva gresit pentru a ne asigura ca simularea va porni;*
- *Apasam butonul de start;*
- *Simulatorul va efectua validarea datelor, introduse de catre utilizator;*
- *In cazul in care datele sunt valide, se porneste simularea;*
- *Urmarim evolutia cozii dorite;*
- *La finalizarea simularii , putem verifica daca programul face ce trebuie in fisierul logFile.txt;*

Scenariu 2:

- *Se apas butul de run, porneste aplicatia de simulare;*

-
- *Apare GUI-ul;*
- *Introducem datele, dar introducem ceva gresit(sau nu introducem nimic, ori uitam sa intoducem ceva);*
- *Apasam butonul de start*
- *Simulatorul va efectua validarea datelor, introduse de catre utilizator;*
- *Se afiseaza mesajul de erroare "Invalid data", deoarece am introdus date incorecte;*
- *Simularea nu porneste;*
- *Dupa introducerea datelor corecte -> scenariul 1;*

6. Concluzii

Concluzia finala este ca am implementat o aplicatie de simulare , usor de folosit de catre oricine si care in cele din urma se comporta cum ne-am astepta sa se comporte . Referitor la partea de cunostiinte accumulate , am recapitulat si aprofundat mai in detaliu aceasta parte a limbajului Java ,anume threaduri ,de asemenea am aprofundat cunostiintele de POO din semestrul trecut , aprofundand la capitolele de GUI , MVC , sau colectii ,pe langa acestea , am folosit pentru prima data AtomicInteger si BlockingQueue .
La partea de dezvoltare ,as imbunatatii interfata grafica .

7. Bibliografie

1… ……. https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf