

DOCUMENTATIE

TEMA 2

NUME STUDENT: POROJAN MĂDĂLIN MARIAN
GRUPA: 30227

CUPRINS

1. Obiectivul temei.....	3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare	4
3. Proiectare	8
4. Implementare.....	12
5. Rezultate.....	13
6. Concluzii	14
7. Bibliografie	15

1. **Obiectivul temei**

Obiectivul principal:

Tema presupune crearea unei aplicații pentru simularea gestionării a mai multor cozi de clienți dintr-un magazin, prin împartirea clienților conform mai multor strategii, pentru a asigura un timp de așteptare redus pentru realizarea serviciilor lor.

Obiectivele secundare:

- Verificarea input-ului introdus pentru a asigura o legătură unu la unu, între ceea ce este introdus în casetele text, corespunzătoare parametrilor de simulare, și ceea ce se află în memoria aplicației;
- Utilizarea a 2 butoane, unul pentru verificarea input-ului dat de utilizator în cadrul celor 7 casete de text, și pentru a realiza simularea conform parametrilor introduși;
- Generarea clienților în mod aleator, conform datelor introduse, și introducerea lor în cozi conform timpului de sosire, cât și a timpului de realizare a serviciilor pentru clienți;
- Afisarea evoluției simulării în timp real și obținerea unor date importante din simulare;
- Interceptarea tuturor erorilor pentru a evita o situație în care programul se află într-o stare necunoscută;
- Informarea utilizatorului despre erori prin pop-up-uri și coduri de eroare care pot fi depanate mai departe consultând un manual specific pentru erori;
- Informarea utilizatorului cu privire la corectitudinea input-ului introdus de la tastatură prin pop-up-uri;
- Utilizarea unui thread separat pentru fiecare coada, pentru a simula paralelismul vieții reale;

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

Analiza problemei:

Ideea principală în realizarea simulării este de a crea o coadă de așteptare pentru noii clienți sosiți, care va fi umplută cu date generate aleator. Fiecare client din coadă de așteptare intră într-o coadă de realizare a serviciilor sale, doar atunci când timpul acestuia de sosire este același cu timpul curent de simulare. La fiecare moment de timp, timpul de realizare a serviciilor pentru clientul din capul cozii scade, iar când cererea clientului este finalizată, se scoate clientul din capul cozii, pentru a realiza serviciile următorului client.

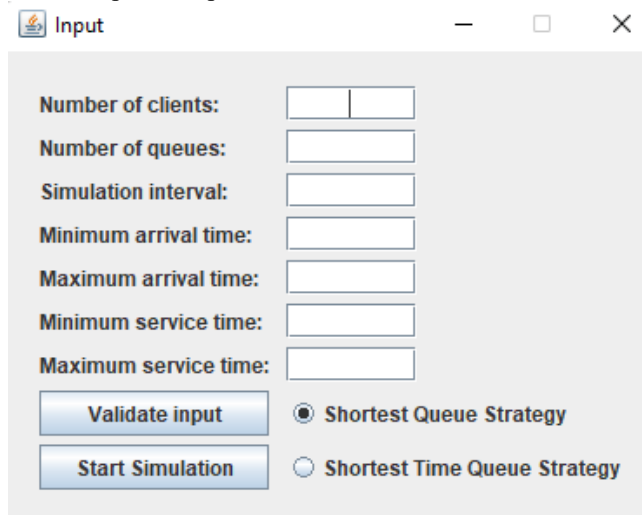
Scenarii posibile:

1. Introducerea datelor de simulare în mod eronat, care ar putea face realizarea simulării imposibile, precum:
 - Introducerea a altor caractere de la tastatură, în casetele text, diferite de cifre;
 - Timpul minim de servicii/sosire este mai mare ca timpul maxim de servicii/sosire;
 - Numarul de cozi este mai mare decât numarul de clienți;
 - Se activează un pop-up ce semnalează ca datele nu sunt introduse corect;
2. Neverificarea input-ului înainte de pornirea simulării:
 - Această problemă este verificată și semnalată printr-un pop-up ce indică apăsarea butonului de validare al input-ului;
3. Neintroducerea parțială a unor date de simulare:
 - În acest caz, este activat un pop-up ce semnalează ca, casetele de text respective sunt goale.

Modelare:

Aplicația propriu-zisă este formată 2 ferestre, una pentru introducerea datelor de către utilizator, și alta pentru realizarea simulării.

Fereastra pentru input:



Este formată din 7 casete text:

- O caseta text pentru introducerea numărului de clienți, „Number of clients”;
- O caseta text pentru introducerea numărului de cozi, „Number of queues”;
- O caseta text pentru introducerea timpului maxim de simulare, „Simulation interval”;
- O caseta text pentru introducerea timpului minim de sosire, pentru clienții generați aleator, „Minimum arrival time”;
- O caseta text pentru introducerea timpului maxim de sosire, pentru clienții generați aleator „Maximum arrival time”;
- O caseta text pentru introducerea timpului minim de realizare a serviciilor, pentru clienții generați aleator, „Minimum arrival time”;
- O caseta text pentru introducerea timpului maxim de realizare a serviciilor, pentru clienții generați aleator „Maximum arrival time”.

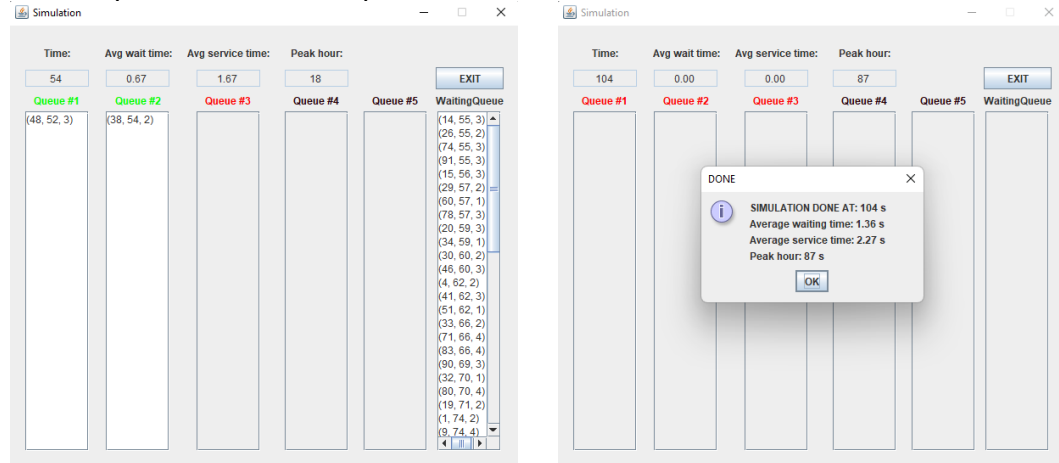
Din 2 radio buttons, folosite pentru a selecta modul în care se realizează împartirea clienților în cozi:

- „Shortest Queue Strategy”;
- „Shortest Time Queue Strategy”.

Cat și din 2 butoane care sunt folosite pentru validarea datelor introduse și pentru a porni simularea:

- „Validate input”;
- „Start simulation”.

Fereastra pentru simularea in timp real:



Este formata din 6 dreptunghiuri pentru afisarea continutului cozilor la un moment de timp. Numele cozilor poate fii **verde**, ce arata ca coada este deschisa si gata sa primeasca noi clienti, **rosu**, ce arata ca coada este goala, dar inca este deschisa, si **rosu inchis**, spre negru, ce arata ca coada este inchisa si nu poate primi niciun client:

- „Queue #1” – arata continutul primei cozi la un moment de timp;
- „Queue #2” – arata continutul a cozii a 2-a la un moment de timp;
- „Queue #3” – arata continutul a cozii a 3-a la un moment de timp;
- „Queue #4” – arata continutul a cozii a 4-a la un moment de timp;
- „Queue #5” – arata continutul a cozii a 5-a la un moment de timp;
- „WaitingQueue” – arata continutul cozii de asteptare la un moment de timp.

Dintr-un buton, „EXIT” folosit pentru a incheia simularea si a iesi din fereastra de simulare.

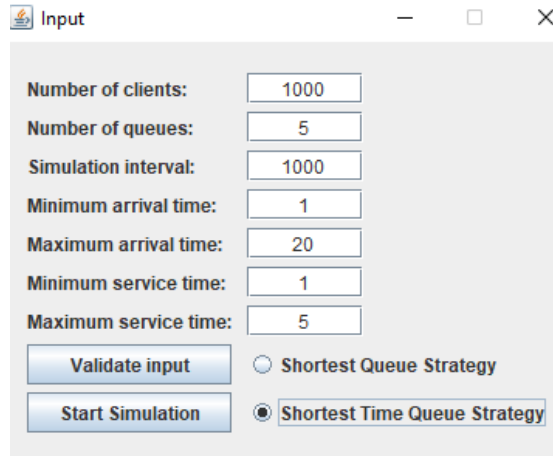
Din 4 casete text:

- „Time” – afiseaza timpul curent;
- „Avg wait time” – afiseaza timpul mediu de asteptare/client;
- „Avg service time” – afiseaza timpul mediu de servicii/client;
- „Peak hour” – afiseaza timpul in care coziile au fost cele mai pline.

La finalizarea simularii, apare un pop-up care informeaza utilizatorul cu privire la timpul mediu de asteptare/client/timp total de simulare si timpul mediu de servicii/client/timp total de simulare, cat si timpul in care coziile au fost cele mai pline si cele mai solicitate.

Cazuri de utilizare:

Înțial se introduc, de la tastatură, drept parametri de testare, 1000 de clienți, 5 cozi, cu un timp de simulare de 1000, un timp minim de arrival de 1, un timp maxim de arrival de 20, un timp minim de service de 1 și un timp maxim de service de 5.



Input

Number of clients: 1000

Number of queues: 5

Simulation interval: 1000

Minimum arrival time: 1

Maximum arrival time: 20

Minimum service time: 1

Maximum service time: 5

Validate input

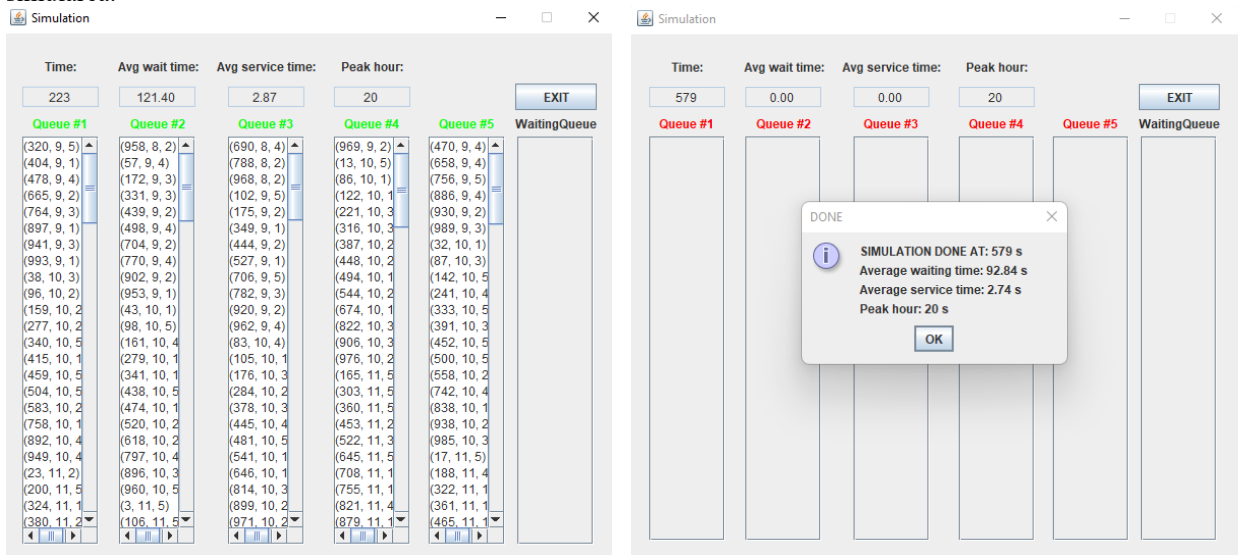
Start Simulation

☐ Shortest Queue Strategy

☒ Shortest Time Queue Strategy

Se alege o strategie, în cazul acesta s-a folosit shortest time strategy.

Pe urmă, se validează input-ul, care este unul corect, și se apasă pe start simulation, pentru a începe simularea.



Simularea s-a finalizat la 579 de secunde, cu un average waiting time de 92.84 s și un average service time de 2.74 s, având un peak hour la 20 de s.

3. Proiectare

Implementarea OOP a aplicației propriu-zise constă în utilizarea modelului MVC (Model-View-Controller), pentru o proiectare concisă și o structurare a tuturor datelor în mod uniform.

1. Pachetul **model** cuprinde doua clase, **Server** si **Task**. Aceste clase vor fi folosite pentru simularea propriu-zisa.
 - Clasa **Task** este defapt clientul care va ajunge in coada. Aceasta clasa contine:
 - Campuri private:
 - „**id**” – **int** ce reprezinta id-ul unic prin care se identifica fiecare client;
 - „**arrivalTime**” – **int** ce reprezinta timpul de sosire al clientului;
 - „**serviceTime**” – **int** ce reprezinta timpul de service al clientului.
 - Accesoare.
 - Clasa **Server** este defapt coada care va fii folosita in simulare drept model. Aceasta coada este formata din mai multe task-uri (clienti). Clasa contine:
 - Campuri private:
 - „**totalServiceTime, waitingPeriod**” – **AtomicInteger** ce reprezinta timpul de asteptare actualizat la fiecare task adaugat;
 - „**tasks**” – care reprezinta un **ArrayBlockingQueue** ce contine doar obiecte de tip **Task**.
 - Metodele, dintre care unele vor fii descrise ulterior:
 - **public Server(int maxTasks);**
 - **public void addTask(Task newTask);**
 - **public synchronized void run();**
 - Accesoare.
2. Pachetul **view** conține strict implementarea grafică a aplicației, cu un constructor care inițializeaza dimensiunile ferestrei, butoanele, cât și denumirea ferestrei și amplaseaza casetele text și butoanele la coordonatele lor date din interfața swing, contine clasele, **InputFrame** si **SimulationFrame**.
3. Pachetul **controller** leagă pachetul **View** de **Controller** și conține clasa care implementează legătura dintre butoane și metodele din codul sursă specifice pentru operațiile pe polinoame, conținând:
 - Clasa **CustomArrayList**, utilizata pentru a suprascrie metoda toString din colectia ArrayList;
 - Clasa **CustomFormatter**, utilizata pentru a formata textul generat de catre logger;
 - Clasa **Scheduler**, care este un planificator al task-urilor, ce determina cum anume vor fii pozitionati clientii in cozi la anumite momente de timp. Un obiect al acestei clase da start la cate un thread corespunzator fiecarui queue, care se va ocupa cu introducerea clientului in coada respectiva. Clasa este formata din:
 - Campuri private:
 - „**maxNoServers**” – **int** reprezinta numarul maxim de cozi care vor fi folosite;
 - „**maxTasksPerServer**” – **int** reprezinta numarul maxim de clienti/coada;
 - „**avgWaitingTime**” – **float** reprezinta timpul mediu de asteptare a unui client, la fiecare moment de timp;
 - „**maxWaitingTime**” – **float** reprezinta timpul maxim de asteptare a unui client, la fiecare moment de timp;
 - „**finalAvgWaitingTime**” – **float** reprezinta timpul mediu de asteptare a unui client, raportat la timpul de simulare;
 - „**finalAvgServiceTime**” – **float** reprezinta timpul mediu de onorare a serviciilor unui client, raportat la timpul de simulare;
 - „**peakHour**” – **int** reprezinta timpul la care cozile au fost cele mai pline;
 - „**servers**” – reprezinta o **lista** ce are ca obiecte fiecare coada utilizata de simulare;
 - „**strategy**” – reprezinta **strategia** folosita in aranjarea clientilor in cozi.
 - Metodele, dintre care unele vor fii descrise ulterior:
 - **public Scheduler(int maxNoServers, int maxTasksPerServer, SelectionPolicy selectionPolicy);**

- public void **changeStrategy**(SelectionPolicy selectionPolicy);
 - public void **dispatchTask**(Task task);
 - public boolean **areServersEmpty**();
 - public float **getAvgWaitingTime** ();
 - public float **getAvgServiceTimePerQueues**();
 - public int **getPeakHour**(int currentTime);
- Clasa enum **SelectionPolicy**, cu **SHORTEST_QUEUE** si **SHORTEST_TIME**;
- Interfata **Strategy**, folosita pentru implementarea independenta a metodei:
 - public void addTask(List<Server> servers, Task task);
- Clasa **ShortestQueueStrategy**, implementeaza **Strategy**, folosita pentru introducerea clientilor intr-o coada gestionata de un thread anume, urmand strategia: clientul pleaca spre coada care are numarul minim de clienti la momentul respectiv.
- Clasa **TimeStrategy**, implementeaza **Strategy**, folosita pentru introducerea clientilor intr-o coada gestionata de un thread anume, urmand strategia: clientul pleaca spre coada care are timpul minim de asteptare la momentul respectiv.
- Clasa **SimulationManager**, este folosita pentru a integra interfata grafica cat si pentru a da un semnal de start catre scheduler, in cazul in care input-ul furnizat este corect, in urma verificarii. In momentul in care utilizatorul apasa pe butonul de „Start simulation”, daca input-ul este corect, se incepe executia unui thread ce gestioneaza interfata grafica, pentru a face posibila observarea evolutiei cozilor in timp real. Clasa este formata din:
 - Campuri private:
 - „isInputFine” – boolean este folosit pentru a determina daca input-ul este sau nu corect;
 - „numberOfClients” – int este numarul de clienti maxim introdus de la tastatura, care urmeaza a fi generati;
 - „numberOfQueues” – int reprezinta numarul de cozi maxime;
 - „simulationInterval” – int reprezinta timpul maxim de simulare;
 - „minArrivalTime” – int reprezinta timpul minim de arrival;
 - „maxArrivalTime” – int reprezinta timpul maxim de arrival;
 - „minServiceTime” – int reprezinta timpul minim de service;
 - „maxServiceTime” – int reprezinta timpul maxim de service;
 - „scheduler” – Scheduler;
 - „inputFrame” – InputFrame;
 - „simulationFrame” – SimulationFrame;
 - „tasks” – List<Task>;
 - „selectionPolicy” – SelectionPolicy;
 - „logger” – Logger;
 - Metodele, dintre care unele vor fii descrise ulterior:
 - public **SimulationManager**();
 - public boolean **verifyAndParseInput**();
 - public List<Task> **generateRandomTasks**();
 - public void **setUpLogger**();
 - public void **updateSimulationFrame**(int currentTime);
 - public void **run**();
 - public static void **main**(String[] args).

Diagrama UML a pachetelor:

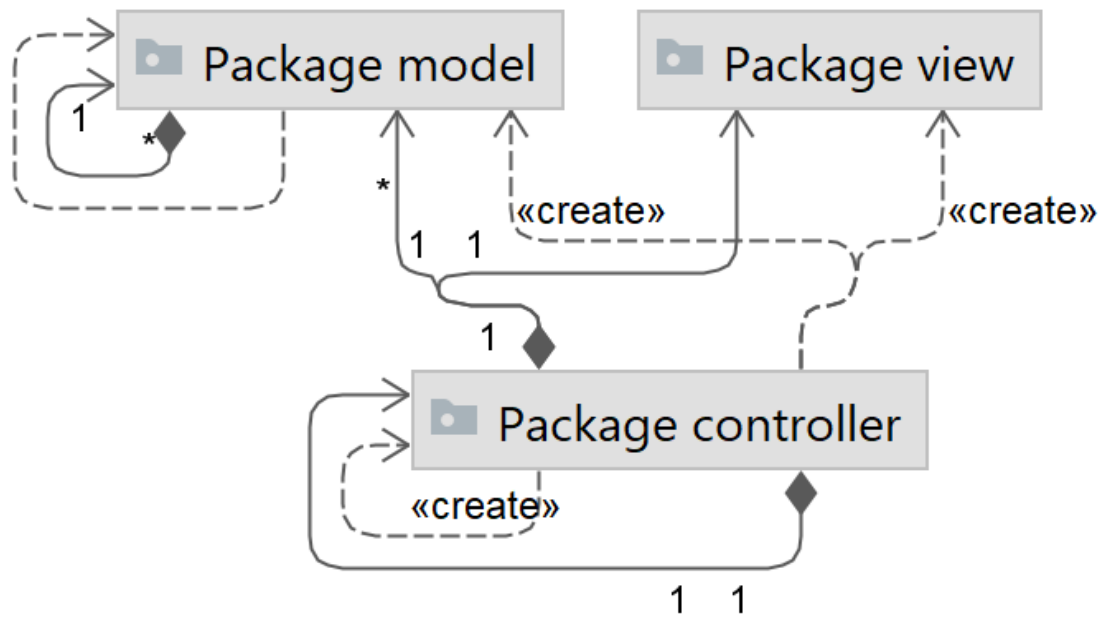
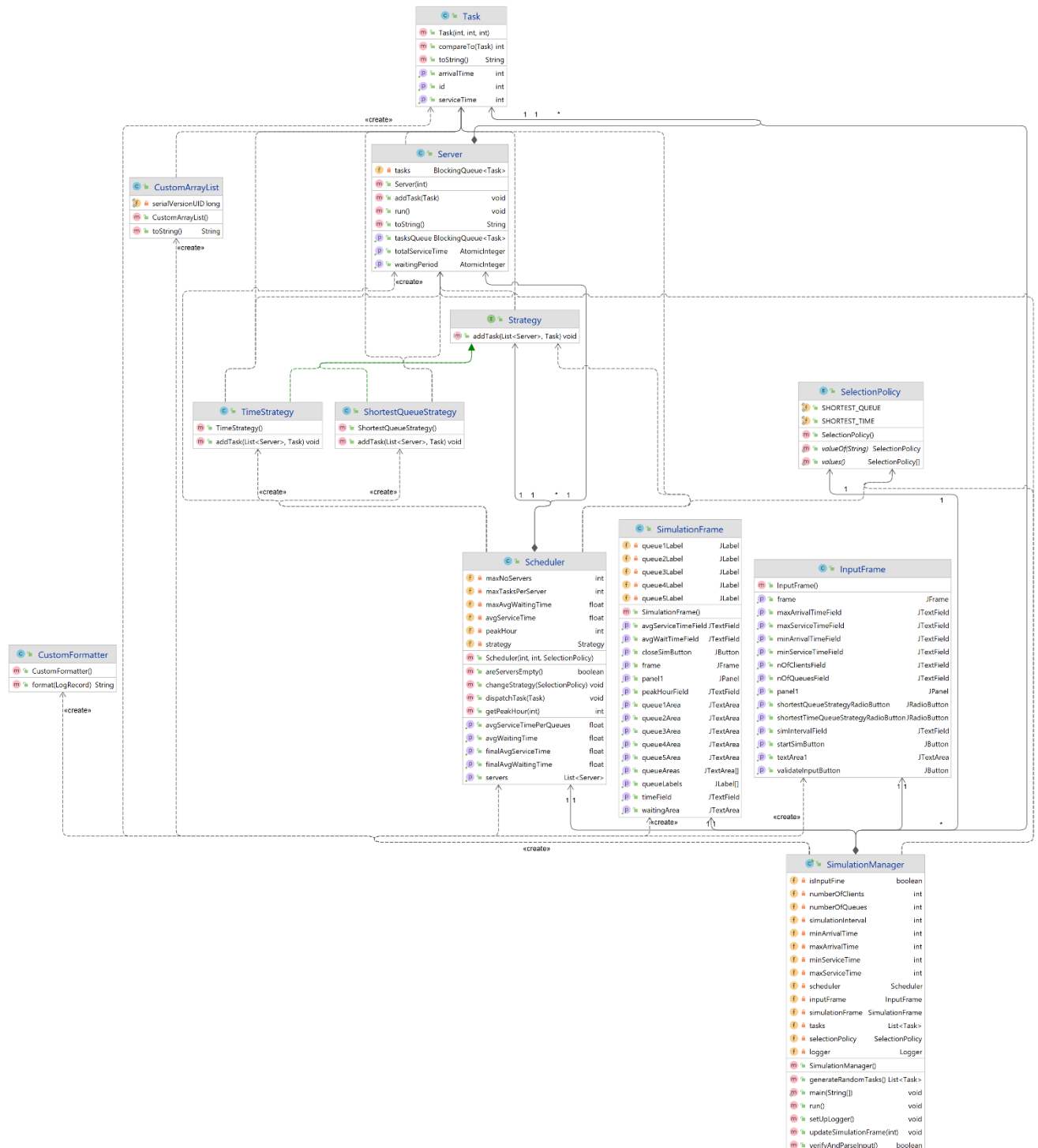


Diagrama UML a claselor:



4. Implementare

Descrierea metodelor importante din unele clase:

- În clasa **Server**:
 - Metoda **run()** este folosită pentru a gestiona thread-urile pornite din constructorul clasei **Scheduler** astfel: parcurgem coada, până când size-ul ei este 0. Thread-ul curent se suspendă pentru un interval de timp de service time înmulțit cu unitatea de măsurare a timpului utilizată. După ce thread-ul iese din sleep, se scoate task-ul din capul queue-ului și se scade perioada de așteptare pentru următoarea persoană.
- În clasa **Scheduler**:
 - Constructorul acestei clase inițializează **maxNoServers** queues, la care le atasează și porneste câte un thread, pentru a asigura paralelismul cozilor.
 - Metoda **changeStrategy** este folosită doar pentru a schimba strategia de adăugare a clienților în cozi, strategie aleasă de către utilizator și furnizată prin constructorul clasei, apelat în clasa **SimulationManager**.
 - Metoda **dispatchTask** dă un client mai departe, la metoda de **addTask** din clasele **ShortestQueueStrategy** sau **TimeStrategy** în funcție de strategia aleasă.
- În clasa **ShortestQueueStrategy** metoda implementată, **addTask**, adaugă un task în queue-ul cu cei mai puțini clienți la coadă.
- În clasa **TimeStrategy** metoda implementată, **addTask**, adaugă un task în queue-ul cu cel mai puțin timp de așteptare.
- În clasa **SimulationManager**:
 - Metoda **generateRandomTasks** generează **numberOfClients** clienți, într-un mod aleator, fiind folosită clasa **Random** din Java. Id-ul fiecărui client este un număr unic, de la 0 la **numberOfClients** – 1 iar **arrivalTime** și **serviceTime** corespunzătoare fiecărui task, sunt cuprinse între minimul și maximul lor aferente (**intervalele**), introduse prin interfața grafică. La final se realizează sortarea task-urilor după timpul de sosire.
 - Constructorul **SimulationManager** preia input-ul de la utilizator și îl verifică în momentul în care este apăsător butonul de Verify input. Dacă input-ul este corect, variabila **isInputFine** devine **true**. Când această variabilă este true, și se apasă pe butonul de Start simulation, este deschisă o nouă fereastră de simulare, este instantiat obiectul clasei **Scheduler**, pornindu-se **numberOfQueues** thread-uri, la finalul acestuia fiind generate cele **numberOfClients** task-uri în mod aleator.
 - Metoda **run()** este folosită pentru a afișa în timp real, evoluția celor **numberOfQueues** thread-uri. Pentru a face acest lucru posibil, este creat și activat un nou thread din constructorul clasei **SimulationManager**. În **run()** este folosită variabila **currentTime** pentru a afișa timpul curent de rulare. Fiecare task din **tasks** este scos din listă și adăugat într-un queue, prin folosirea clasei **Scheduler**, care decide, în funcție de strategia aleasă, în ce coadă va face inserarea acestui task. Totodată, în această metodă se face un log, corespunzător datelor obținute pe parcursul simulării, fiind folosită clasa **Logger**, pentru a face posibil acest lucru.

5. Rezultate

Pentru a testa aplicatia am folosit testele stabilite pentru acest assignment. Am realizat 3 teste, care se regasesc in folderul **appTesting**, test1.txt, test2.txt si test3.txt.

Primul test cuprinde:

- Number of clients = 4;
- Number of queues = 2;
- Simulation interval = 60 s;
- [Minimum arrival time, Maximum arrival time] = [2, 30];
- [Minimum service time, Maximum service time] = [2, 4].
- Cu rezultatele:
 - Avg waiting time: 0.30 s
 - Avg service time: 1.02 s
 - Peak Hour: 17 s.

Al doilea test cuprinde:

- Number of clients = 50;
- Number of queues = 5;
- Simulation interval = 60 s;
- [Minimum arrival time, Maximum arrival time] = [2, 40];
- [Minimum service time, Maximum service time] = [1, 7].
- Cu rezultatele:
 - Avg waiting time: 1.50 s
 - Avg service time: 4.04 s
 - Peak Hour: 31 s

Al treilea test cuprinde:

- Number of clients = 1000;
- Number of queues = 20;
- Simulation interval = 200 s;
- [Minimum arrival time, Maximum arrival time] = [10, 100];
- [Minimum service time, Maximum service time] = [3, 9].
- Cu rezultatele:
 - Avg waiting time: 20.53 s
 - Avg service time: 5.74 s
 - Peak Hour: 100 s

6. Concluzii

În concluzie, această tema mi-a dezvoltat cunoștințele de OOP, învățând să lucrez cu thread-uri pe care le voi utiliza în viitoarele proiecte, dovedindu-se un aspect foarte important pentru împărțirea clientilor în cozi. În același timp, mi-am dezvoltat abilitățile de structurare și depanare a codului, utilizând platforma Gitlab ce s-a dovedit extrem de folositoare în acest scop.

Ca dezvoltări ulterioare, se poate opta pentru o nouă interfață grafică, mai actuală și mai accesibilă, cu noi funcții, cum ar fi istoric de simulări, istoric de rezultate de simulare, etc.

7. Bibliografie

1. https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf
2. <https://www.thoughtco.com/using-java-naming-conventions-2034199>
3. <https://docs.oracle.com/javase/tutorial/uiswing/>
4. <http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
5. <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
6. http://www.tutorialspoint.com/java/util/timer_schedule_period.html