

TEHNICI DE PROGRAMARE FUNDAMENTALE
ASSIGNMENT 2

QUEUES SIMULATOR
DOCUMENTATIE

Kovacs Alexandru
Grupa 30223

Cuprins

1. Obiectivul temei.....	3
1.1. Obiectivul principal	3
1.2. Obiective secundare	3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare	3
2.1. Analiza problemei	3
2.2. Cazuri de utilizare	4
3. Proiectare	5
3.1. Diagrama de pachete	5
3.2. Diagrama de clase	6
3.3. Structuri de date folosite.....	9
3.3.1. ArrayBlockingQueue	9
3.3.2. CopyOnWriteArrayList	9
3.3.3. AtomicInteger	9
3.3.4. Thread	9
3.3.5. ExecutorService.....	9
3.4. Interfete definite	9
3.4.1. Strategy	9
3.4.2. Logger.....	9
3.4.3. SimulationConfiguraion	9
4. Implementare	10
4.1. Descriere clase.....	10
4.2. Descriere implementare interfata utilizator	15
4.2.1. Fereastra principala	16
4.2.2. Fereastra de stari	17
4.2.3. Fereastra de vizualizare	17
5. Rezultate	18
6. Concluzii	20
7. Bibliografie	20

1. Obiectivul temei

1.1. Obiectivul principal

Obiectivul principal al acestei teme de laborator este de a proiecta si a implementa o aplicatie care permite utilizatorului simulareacomportamentului unor cozi. Pentru a porni o astfel de simulare, utilizatorul trebuie sa introduca mai intai parametri doriti prin intermediul interfetei grafice incluse in aplicatie. Apoi utilizatorul poate lansa una sau mai multe simulari apasand pe butonul „Start Simulation!”. Acesta va avea acces la date in timp real privind starea simularii prin intermediul unei noi ferestre din interfata grafica. Rezultatele simularii vor fi salvate dupa finele acesteia intr-un fisier text.

1.2. Obiective secundare

Obiectivele secundare care vor fi abordate in aceasta tema sunt urmatoarele:

- Analiza problemei si identificarea cerintelor – **Capitolul 2**
- Proiectarea simulatorului de cozi – **Capitolul 3**
- Implementarea simulatorului de cozi – **Capitolul 4**
- Testarea simulatorului de cozi – **Capitolul 5**

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

2.1. Analiza problemei

Cerintele functionale care reies din analiza enuntului temei de laborator sunt:

- Simulatorul de cozi ar trebui sa permita utilizatorilor sa introduca parametri de simulare
- Simulatorul de cozi ar trebui sa permita alegerea unei strategii dintre cele specificate:
 - Shortest Queue Strategy
 - Shortest Time Strategy
- Simulatorul de cozi ar trebui sa permita utilizatorilor sa porneasca una sau mai multe simulari
- Simulatorul de cozi ar trebui sa afiseze rezultatul simularii pornite atat in interfata grafica cat si in fisierul text
- Simulatorul de cozi ar trebui sa notifice utilizator in cazul aparitiei unei erori

2.2. Cazuri de utilizare

Caz de utilizare: pornirea unei simulari

Actorul principal: utilizatorul

Scenariul principal:

1. Utilizatorul introduce parametri doriti pentru simulare utilizand interfata grafica.
2. Utilizatorul apasa butonul „Start Simulation!” pentru a incepe simularea.
3. Se vor deschide doua ferestre noi pentru observarea simularii in timp real.
4. La finele simularii se va afisa valoarea „Finished” in casuta de stare gasita in prima fereastră noua deschisa si se va salva intr-un fisier text log-urile simularii.
5. Acum utilizatorul poate inchide ferestrele noi deschise si sa efectueze alta simulare.

Secventa alternativa:

- In eventualitatea aparitiei unei erori, aceasta este afisata utilizatorului.
- Scenariul ajunge la pasul 1.

Caz de utilizare: pornirea simultana a mai multor simulari

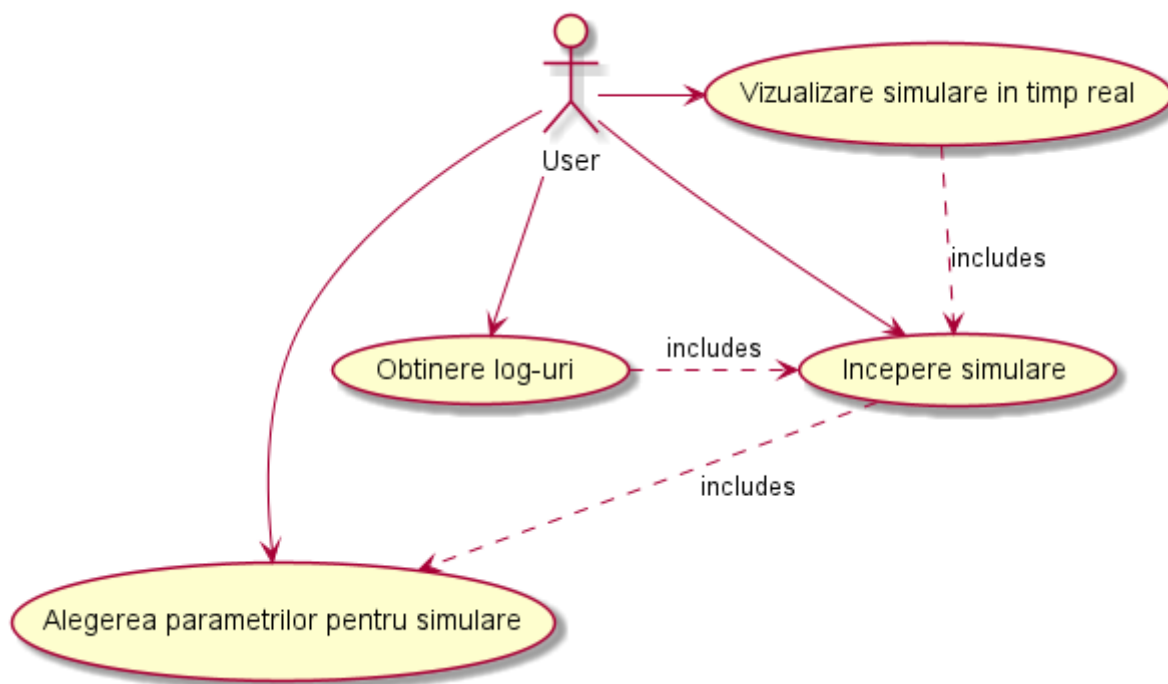
Actorul principal: utilizatorul

Scenariul principal:

1. Utilizatorul introduce parametri doriti pentru simulare utilizand interfata grafica.
2. Utilizatorul apasa butonul „Start Simulation!” pentru a incepe simularea.
3. Se repeta pasii 1 – 2 de cate ori se doreste.
4. Se vor deschide cate doua ferestre noi pentru fiecare simulare pornita.
5. La finele fiecarei simulari se va afisa valoarea „Finished” in casuta de stare gasita in fereastră noua deschisa pentru simularea respectiva si se va salva intr-un fisier text log-urile simularii terminate.
6. Acum utilizatorul poate inchide ferestrele noi deschise si sa efectueze alta simulare.

Secventa alternativa:

- In eventualitatea aparitiei unei erori, aceasta este afisata utilizatorului.
- Scenariul ajunge la pasul 1.

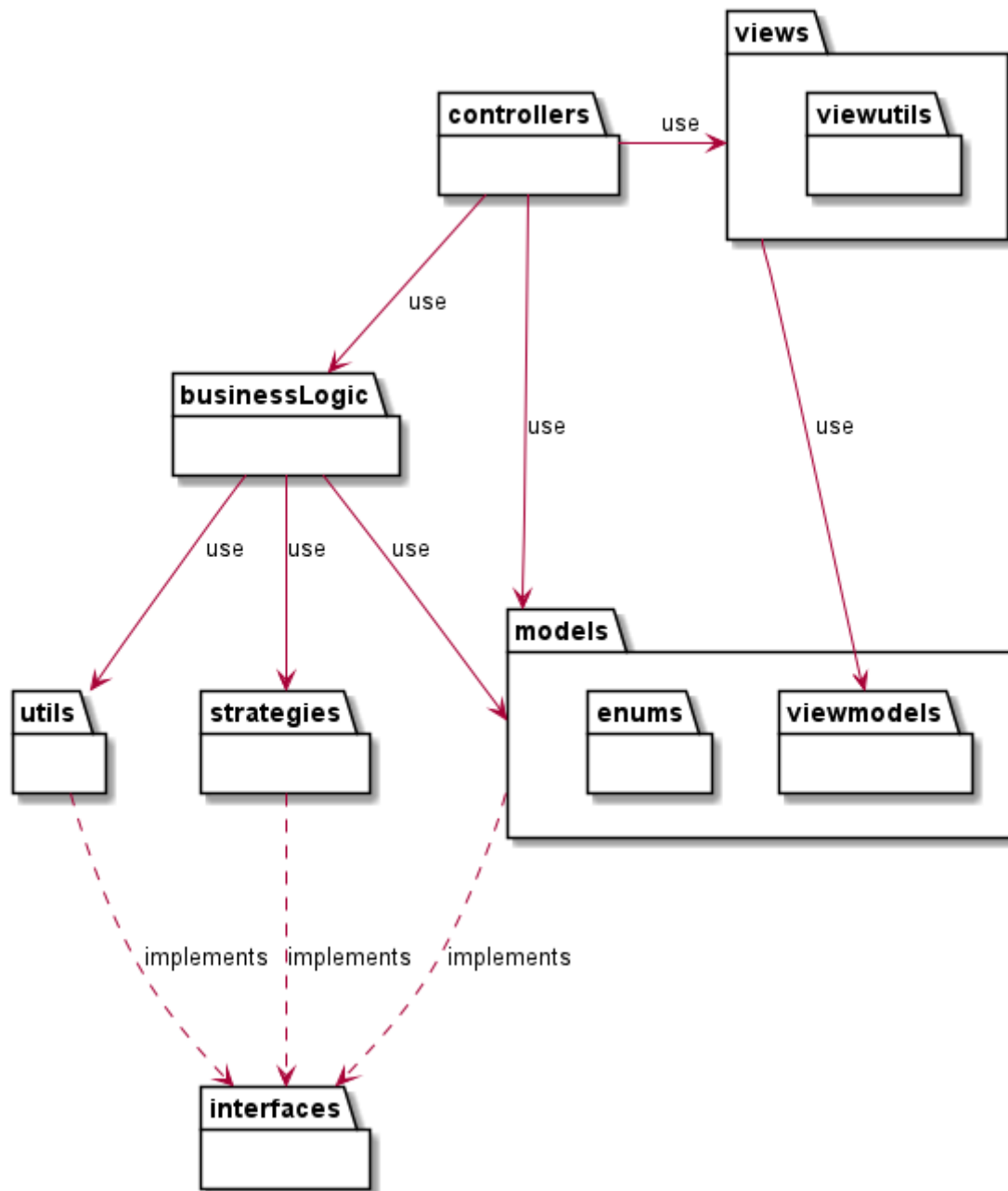


3. Proiectare

3.1. *Diagrama de pachete*

Am conceput acest proiect urmarind structura de pachete prezentata in figura de mai jos.

Se poate observa ca am utilizat un model architectural de tip MVC (Model View Controller) alaturi de un model de proiectare bazat pe strategie.

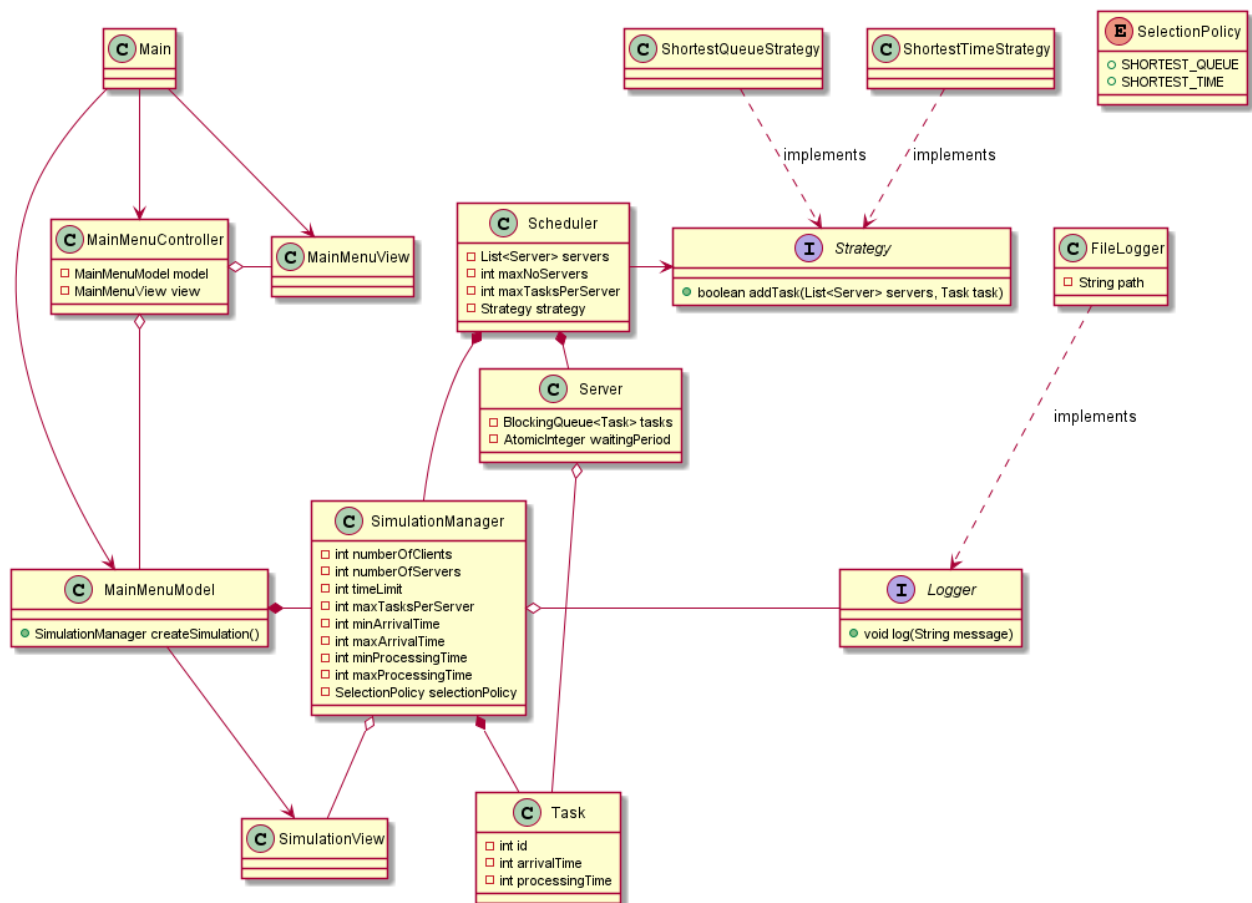


Pachetele folosite sunt:

- **Controllers**
 - contine toate controllerele folosite in arhitectura MVC
- **Views**
 - contine toate view-urile folosite in arhitectura MVC
- Utils
 - contine toate clasele care ajuta la crearea / initierea componentelor UI

- **Models**
 - contine toate clasele care memoreaza date si / sau stari folosite in aplicatie
 - Enums
 - contine toate enumerarile folosite de modelele
 - ViewModels
 - contine toate clasele folosite in view-uri care retin starea aplicatiei
- **BusinessLogic**
 - contine toate clasele de baza de care aplicatia se foloseste pentru a luat decizii
- **Strategies**
 - contine toate strategiile utilizate in aplicatie
- **Utils**
 - contine toate clasele cu functionalitate de sine statatoare utilizate in restul proiectului
- **Interfaces**
 - contine toate interfetele folosite in intregul proiect

3.2. Diagrama de clase

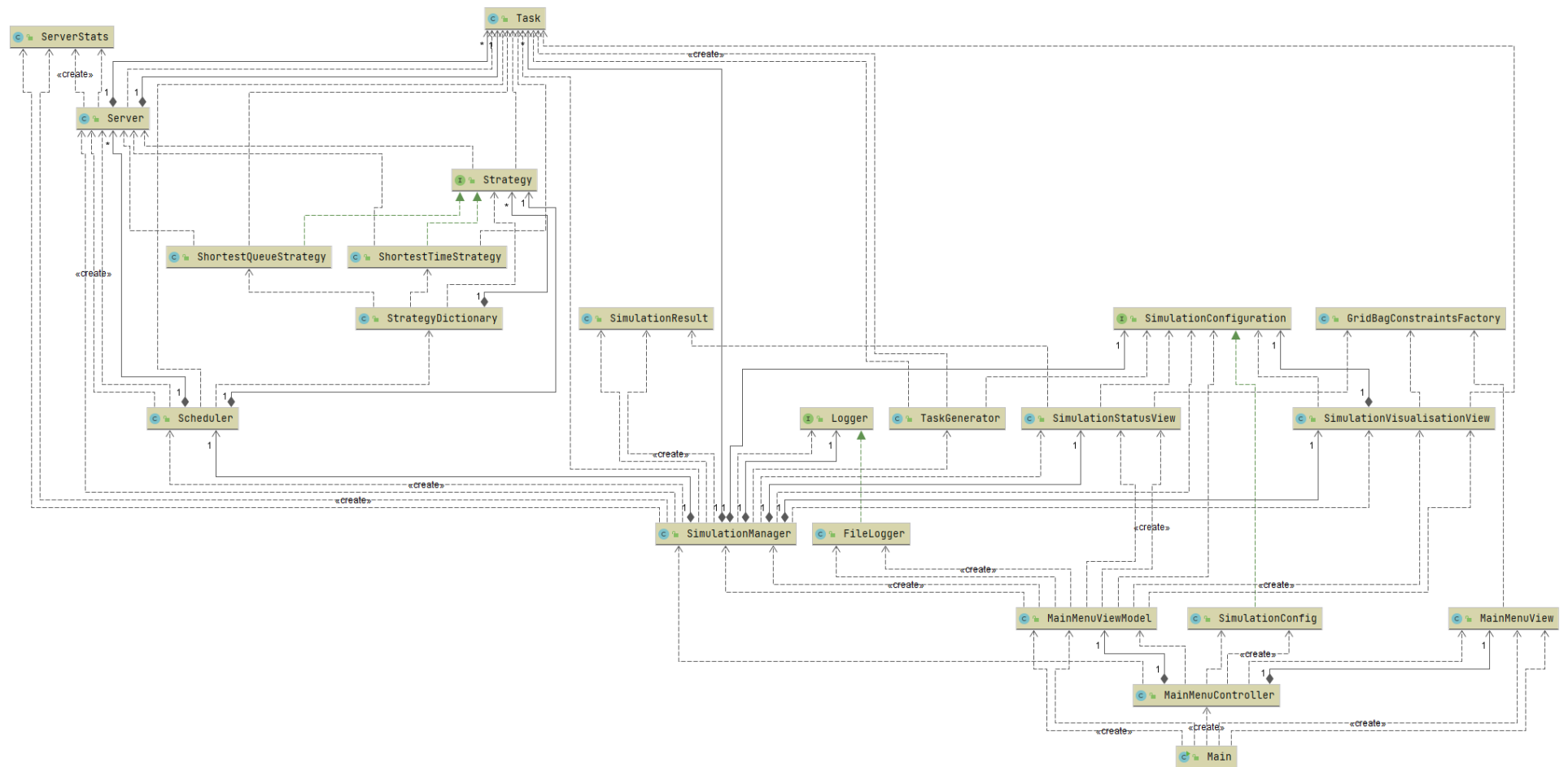


Claase identificate in timpul procesului de proiectare a acestei teme de laborator sunt:

- **Main**
 - este utilizata la crearea claselor principale (Model, View si Controller) folosite in arhitectura MVC
- **MainMenuModel**
 - este modelul folosit de catre controller pentru a retine starea curenta a aplicatiei si pentru a crea noi simulari impreuna cu view-urile corespunzatoare acestora
- **MainMenuView**

- este utilizata la construirea interfetei grafice dedicate si permite utilizatorului selectarea parametrilor de simulare, cat si inceperea unei simulari
- **MainMenuController**
 - este utilizata pentru a intercepta evenimentele cauzate de catre utilizator si pentru a modifica starea curenta a aplicatiei
- **SimulationManager**
 - este utilizata pentru logica si procesarea unei simulari
- **Scheduler**
 - este utilizata pentru intretinerea thread-urilor cozilor simulate
- **Task**
 - este utilizata pentru a retine id-ul unui client, timpul acestuia de sosire si timpul acestuia de procesare
- **SimulationView**
 - este utilizata la vizualizarea starii si rezultatelor simularii
- **Strategy**
 - este o interfata utilizata pentru a imparti clientii la cozi in functie de o regula prestabilita
- **ShortestTimeStrategy**
 - este utilizata pentru a selecta coada cu cel mai mic timp de asteptare pentru a fi introdus un nou client
- **ShortestQueueStrategy**
 - este utilizata pentru a selecta coada cu cei mai putini clienti la rand pentru a fi introdus un nou client
- **Logger**
 - este o interfata utilizata pentru a salva un log
- **FileLogger**
 - este utilizata pentru a salva un log intr-un fisier text
- **SelectionPolicy**
 - este o enumeratie care specifica toate strategiile definite in cadrul aplicatiei noastre

Aici avem diagrama de clase rezultata la finalul implementarii proiectului:



Powered by yFiles

Interfețele definite sunt Strategy, Logger și SimulationConfiguration. Aceste interfețe au rolul atât de a desprinde funcționalitatea de obiectul înșiși, cât și pentru a ascunde unele atribute nedorite ale obiectelor folosite.

3.3. Structuri de date folosite

3.3.1. ArrayBlockingQueue

Structura de date ArrayBlockingQueue este o structura de date thread-safe atat pentru operatiile de citire, cat si pentru operatiile de scriere care ne ofera functionalitatea unei cozi.

Am folosit structura de date ArrayBlockingQueue pentru a simula modul de asteptare al clientilor intr-o coada in fiecare clasa Server creata.

3.3.2. CopyOnWriteArrayList

Structura de date CopyOnWriteArrayList este o structura de date thread-safe atat pentru operatiile de citire, cat si pentru operatiile de scriere care ne ofera functionalitatea unei liste.

Am folosit structura de date CopyOnWriteArrayList pentru a partaja informatii intre thread-ul principal al simularii si cele secundare (serverele).

3.3.3. AtomicInteger

Structura de date AtomicInteger este o structura de date thread-safe atat pentru operatiile de citire, cat si pentru operatiile de scriere care ne ofera functionalitatea unei variabile de tip integer.

Am folosit structura de date AtomicInteger pentru a partaja informatii intre thread-ul principal al simularii si cele secundare (serverele).

3.3.4. Thread

Structura de date Thread este o structura de date care ne ajuta sa ne folosim de capacitatea procesorului de a efectua sarcini in mod concurrent.

Am folosit aceasta structura pentru a separa logica unei simulari de interfata grafica.

3.3.5. ExecutorService

Structura de date de tip Executorservice este o structura de care ne usureaza munca cu thread-uri si ne pune la dispozitie un set de metode care faciliteaza administrarea acestora.

Am folosit structura de date de tip ExecutorService pentru a crea thread-uri pentru fiecare coada pe care dorim sa o simulam. Am folosit aceasta structura si pentru inchiderea acestora.

3.4. Interfete definite

3.4.1. Strategy

Interfata Strategy are rol de a abstractiza logica de a alege coada la care urmeaza sa fie pus un client de implementarea acesteia.

3.4.2. Logger

Interfata Logger are rol de a abstractiza logica de a salva niste log-uri de implementarea acesteia.

3.4.3. SimulationConfiguraion

Interfata SimulationConfiguraion are rol de a permite accesul doar la citirea datelor din obiectele care o implementeaza..

4. Implementare

4.1. Descriere clase

Clasa MainMenuController

- Este componenta care gestioneaza interactiunea utilizatorului cu aplicatia si lucreaza cu modelul.
- Este C-ul din modelul arhitectural MVC.

`private final MainMenuViewModel model;`

- Obiect ce memoreaza starea curenta a aplicatiei si genereaza instante noi de simulare

`private final MainMenuView view;`

- Obiect ce contine interfata grafica (view-ul)

`private void setUpUserEvents()`

- Metoda ce atribuie componentelor din interfata grafica ascultatoare de evenimente (ActionListener)

Interfata Strategy

- Este o interfata folosita pentru a separa logica de a distribui un client de implementarea acesteia.

Interfata Logger

- Este o interfata folosita pentru a separa logica de a salva un set de log-uri de implementarea acesteia.

Interfata SimulationConfiguration

- Este o interfata folosita pentru a asigura imutabilitatea obiectului ce contine date despre parametrii de simulare.

Enumeratia SelectionPolicy

- Este o enumeratie care contine toate strategiile de a distribui un client la o coada definite in aplicatia noastra.

Clasa SimulationManager

- Este o clasa care contine logica si se ocupa de intretinerea unei simulari.

`private final SimulationConfiguration config;`

- Este un obiect ce contine parametrii pentru simularea curenta.

`private final Scheduler scheduler;`

- Este un obiect ce se ocupa de intretinerea cozilor din simularea curenta.

`private final SimulationStatusView view;`

- Este un obiect ce contine interfata grafica unde se vor afisa informatii despre simularea curenta.

`private final SimulationVisualisationView secondaryView;`

- Este un obiect ce contine interfata grafica unde se va face vizualizarea simularii curente.

`private final Logger logger;`

- Este un obiect ce se ocupa de salvarea log-urilor.

`private final List<Task> generatedTasks;`

- Este o lista ce memoreaza clientii care inca nu au ajuns la o coada.

`private void dispatchClients(int currTime)`

- Metoda ce trimite clienti la scheduler in vederea distribuirii lor la cozi.

`private void updateUI(SimulationResult result)`

- Metoda ce imparte doua monoame.

`private void log(String log)`

- Metoda ce distribuie log-ul dorit..

`@Override`

`public void run()`

- Metoda ce se ocupa cu rularea simularii.

Clasa Scheduler

- Este o clasa care se ocupa de intretinerea thread-urilor pe care sunt simulate cozile si de distribuirea clientilor.

`private final ExecutorService executor;`

- Obiect ce se ocupa de crearea si terminarea thread-urilor.

`private final List<Server> servers;`

- Lista contine toate cozile care vor fi simulate.

`private final int maxNoServers;`

- Variabila ce contine numarul de cozi simulate.

`private final int maxTasksPerServer;`

- Variabila ce contine numarul maxim de client dintr-o coada.

`private Strategy strategy;`

- Obiect ce contine logica de distribuire a unui client la o coada.

`public void changeStrategy(SelectionPolicy policy)`

- Metoda ce schimba strategia de distribuire a clientilor in cozi.

`public boolean dispatchTask(Task t)`

- Metoda ce distribuie un client intr-o coada in functie de strategia selectata.

`public void shutdown()`

- Metoda ce opreste thread-urile pe care sunt simulate cozile.

Clasa Server

- Este o clasa care contine logica unei cozi simulate.

`private final BlockingQueue<Task> tasks;`

- Coada ce simuleaza clientii stand la coada.

`private final AtomicInteger addedNumber;`

- Variabila ce indica numarul de clienti adaugat la coada de la ultima verificare.

`private final AtomicInteger waitingPeriod;`

- Variabila ce memoreaza timpul total de asteptat la coada respectiva.

`private final AtomicInteger waitedTime;`

- Variabila ce memoreaza numarul de unitati de timp asteptate la coada respectiva.

`private final AtomicInteger waitedTasks;`

- Variabila ce memoreaza numarul de clienti care au asteptat la coada respectiva.

private final AtomicInteger **processedTime**;

- Variabila ce memoreaza numarul de unitati de timp procesate la coada respectiva.

private final AtomicInteger **processedTasks**;

- Variabila ce memoreaza numarul de clienti care au fost procesati la coada respectiva.

private Task **currTask**;

- Obiect ce memoreaza clientul care se proceseaza in momentul de fata.

public boolean **addTask**(Task newTask)

- Metoda ce adauga un client la coada. Returneaza true in caz de succes si false altfel.

@Override

public void **run**()

- Metoda ce se ocupa cu rularea simularii pe coada respectiva.

@Override

public String **toString**()

- Metoda folosita la transpunerea clientilor din coada in format text.

Clasa MainMenuViewModel

- Este componenta care retine starea aplicatiei si este folosita pentru crearea unor noi instante de simulare.
- Este M-ul din modelul arhitectural MVC.

private String **path** = ".\\";

- Variabila ce retine calea unde vor fi salvate log-urile .

private int **simulationNb**;

- Variabila ce retine cate simulari au fost facute.

public SimulationManager **createSimulation**(SimulationConfiguration config)

- Metoda ce reseteaza starea aplicatiei si textul care urmeaza sa fie afisat utilizatorului.

Clasa Task

- Este un model care retine informatii despre un client.

private int **id**;

- Variabila ce retine id-ul unui client.

private int **arrivalTime**;

- Variabila ce retine timpul de sosire al unui client.

private int **processingTime**;

- Variabila ce retine timpul de procesare al unui client.

@Override

public String **toString**()

- Metoda folosita la transpunerea clientului in format text.

Clasa ServerStats

- Este un model care retine informatii despre un server.

Clasa SimulationConfig

- Este un model care retine parametrii pentru simulare.

```
private int numberOfClients;  
private int numberOfServers;  
private int timeLimit;  
private int maxTasksPerServer;  
private int minArrivalTime;  
private int maxArrivalTime;  
private int minProcessingTime;  
private int maxProcessingTime;  
private SelectionPolicy selectionPolicy;
```

Clasa SimulationConfig

- Este un model care retine rezultatele simularii.

```
private double avgWaitTime;  
private double avgProcessTime;  
private int peakHour;
```

Clasa ShortestTimeStrategy & Clasa ShortestQueueStrategy

- Este o clasa ce implementeaza logica de distribuire a unui client intr-o coada.

@Override

```
public boolean addTask(List<Server> servers, Task task)
```

- Metoda care adauga clientul intr-una din cozile specificate in functie de o regula predefinita.

Clasa FileLogger

- Este o clasa care implementeaza logica de salvare a unor log-uri. Salvarea se face intr-un fisier text.

@Override

```
public void log(String message)
```

- Metoda care salveaza log-ul intr-un fisier text.

Clasa StrategyDictionary

- Este o clasa statica care faciliteaza obtinerea unei strategii in functie de valoarea SelectionPolicy dorita.

```
static Map<SelectionPolicy, Supplier<Strategy>> map;
```

- Structura de date care contine un SelectionPolicy ca si cheie si un constructor de strategie ca si valoare.

```
public static Strategy get(SelectionPolicy policy)
```

- Metoda care intoarce o instanta de strategie dorita in functie de SelectionPolicy-ul precizat.

Clasa GridBagConstraintsFactory

- Este o clasa statica care faciliteaza obtinerea mai usoara a unor GridBagConstraints cu diferite optiuni aplicate.

Clasa TaskGenerator

- Este o clasa statica care faciliteaza obtinerea unei liste de clienti generati in mod aleatoriu in functie de parametrii specificati de utilizator.

`public static List<Task> generateTasks(SimulationConfiguration config)`

- Metoda care genereaza in mod aleatoriu o lista de clienti folosind parametrii specificati.
- Lista de clienti va fi sortata crescator in functie de timpul de sosire.

Clasa MainMenuView

- Este o componenta care structureaza interfata grafica dedicata a aplicatiei.
- Este V-ul din modelul architectural MVC.

`private void initComponents()`

- Metoda care initializeaza toate componentele necesare pentru acest view.

`private void setUpUI()`

- Metoda care pozitioneaza toate componentele necesare pentru acest view la pozitiile dorite.

`public void showError(String message)`

- Metoda care afiseaza un mesaj de eroare utilizatorului.

`public void addBtnActionListener(ActionListener al)`

- Metoda care atribuie un eveniment butonului „Start Simulation”.

Clasa SimulationStatusView

- Este o componenta care structureaza interfata grafica dedicata a aplicatiei.
- Are rolul de a afisa starea curenta a simularii.

`public void addLog(String log)`

- Metoda care adauga in interfata grafica log-ul dorit.

`public void setResults(SimulationResult result)`

- Metoda care actualizeaza informatiile din interfata grafica cu noile rezultate ale simularii.

Clasa SimulationVisualisationView

- Este o componenta care structureaza interfata grafica dedicata a aplicatiei.
- Are rolul de a afisa starea curenta a simularii sub forma unor imagini.

`public void updateLeftClients(List<Task> clients)`

- Metoda care actualizeaza lista clientilor din interfata grafica care inca nu au ajuns la coada.

`public void updateQueue(int index, List<Task> clients)`

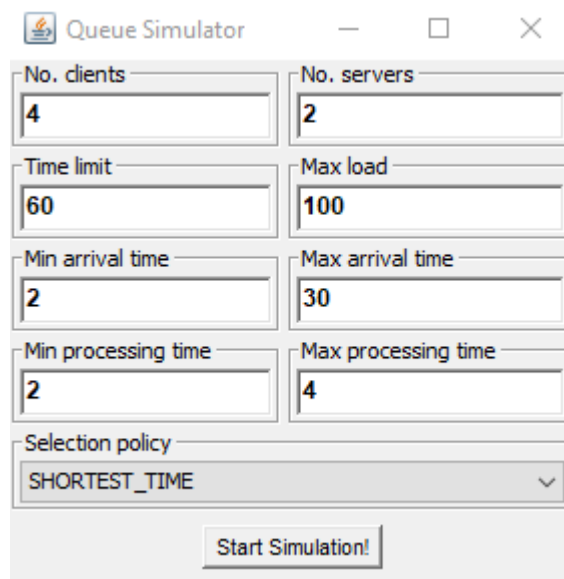
- Metoda care actualizeaza listele cozilor din interfata grafica in functie de noua stare a simularii.

Clasa Main

- Este o clasa statica in care se face initierea componentelor necesare pentru a completa modelul architectural MVC.
- Este punctul de start al aplicatiei.

4.2. Descriere implementare interfata utilizator

Interfata grafica este una user-friendly. Aceasta este usor de folosit, intuitiva si nu permite introducerea de date incorecte.



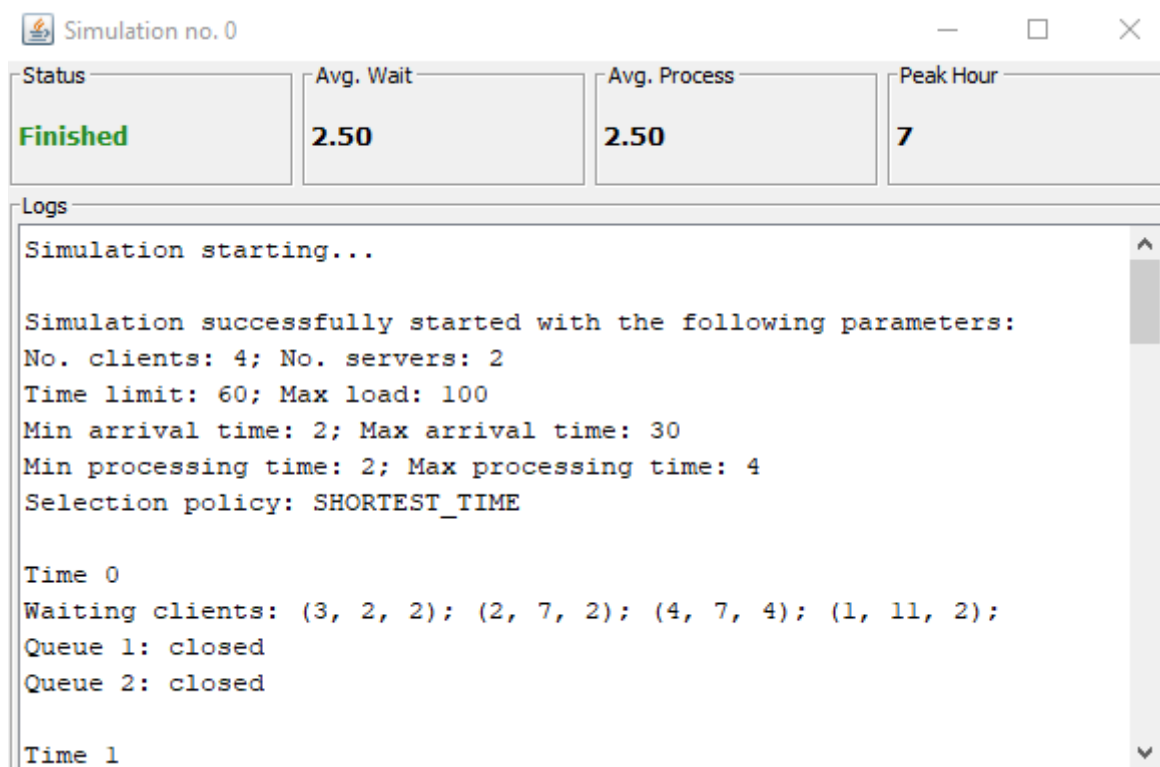
The screenshot shows a window titled "Queue Simulator" with standard Windows window controls. It contains several input fields for simulation parameters:

Parameter	Value
No. clients	4
No. servers	2
Time limit	60
Max load	100
Min arrival time	2
Max arrival time	30
Min processing time	2
Max processing time	4
Selection policy	SHORTEST_TIME

At the bottom of the window is a button labeled "Start Simulation!".

Figura 1

Interfata grafica a aplicatiei este alcatuita din trei JFrame-uri: unul principal ([Figura 1](#)) in care se poate face introducerea parametrilor pentru simulare si pornirea simularii, unul pentru afisarea starii actuale a simularii ([Figura 2](#)) si unul pentru o afisare vizuala (sub forma de poze) a starii actuale a simularii ([Figura 3](#)). Aceste 3 JFrame-uri au structura principala formata din 2 panel-uri principale imbricate. Primului panel i-am atribuit un layout de tip GridLayout care ofera o functionalitate grosiera de redimensionare a componentelor cuprinse in acest JFrame. Al doilea panel, inclus in primul, are atribuit un layout de tip GridBagLayout pentru a putea pozitiona elementele componente dupa propriile preferinte.



The screenshot shows a window titled "Simulation no. 0" with standard Windows window controls. It displays simulation results in a structured layout:

Status	Avg. Wait	Avg. Process	Peak Hour
Finished	2.50	2.50	7

Below the table is a "Logs" section with a scrollable text area containing the following text:

```
Simulation starting...

Simulation successfully started with the following parameters:
No. clients: 4; No. servers: 2
Time limit: 60; Max load: 100
Min arrival time: 2; Max arrival time: 30
Min processing time: 2; Max processing time: 4
Selection policy: SHORTEST_TIME

Time 0
Waiting clients: (3, 2, 2); (2, 7, 2); (4, 7, 4); (1, 11, 2);
Queue 1: closed
Queue 2: closed

Time 1
```

Figura 2

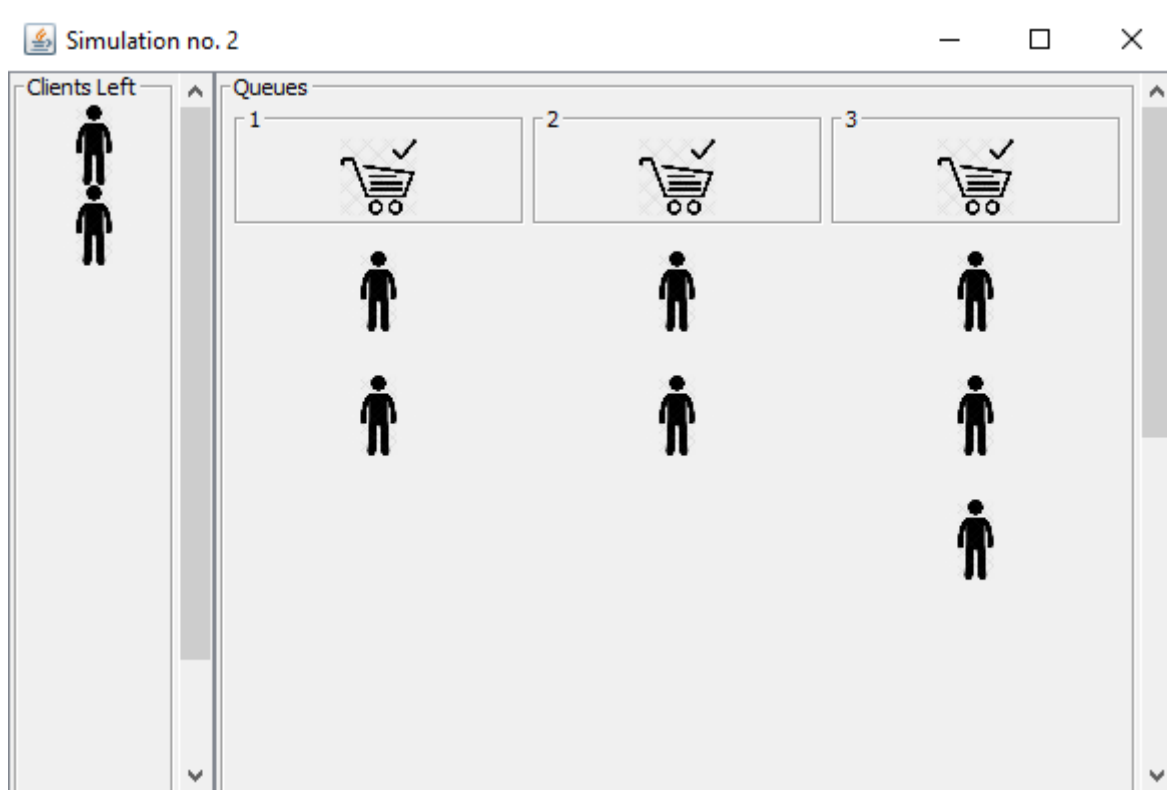


Figura 3

4.2.1. Fereastra principala

Fereastra principala (Figura 1) este formata pe langa cele 2 panel-uri principale mentionate anterior dintr-un numar de 10 alte panel-uri. Fiecare astfel de panel, mai putin ultimul, are setata o bordura de tip `TitleBorder` pentru a afisa campul asupra caruia se vor face modificari folosind panel-ul corespunzator.

Primele 8 panel-uri contin fiecare cate un `TextField` care are rolul de a permite utilizatorului introducerea parametrilor doriti pentru simulare si sunt asezate pe 4 randuri si 2 coloane cu ajutorul layout-ului de tip `GridBagLayout` ai panel-ului in care sunt incluse.

Aceste 8 panel-uri reprezinta, in ordine, urmatoorii parametri:

- Numarul de clienti
- Numarul de cozi
- Numarul de unitati de timp in care se desfasoara simularea
- Timpul minim de sosire al unui client
- Timpul maxim de sosire al unui client
- Timpul minim de procesare al unui client
- Timpul maxim de procesare al unui client

Penultimul panel contine un `JComboBox` in care sunt prezente toate strategiile implementate de catre aplicatie. Acest panel este extins pe doua coloane folosind un obiect de tipul `GridBagConstraints`.

Ultimul panel are un layout de tip `FlowLayout` pentru a centra butonul „Start Simulation!” din interiorul acestuia. Acest panel este, de asemenea, extins pe doua coloane folosind un obiect de tipul `GridBagConstraints`.

4.2.2. Fereastra de stari

Fereastra de stari (**Figura 2**) este formata pe langa cele 2 panel-uri principale mentionate anterior dintr-un numar de 6 alte panel-uri. Fiecare astfel de panel, mai putin ultimul, are setata o bordura de tip `TitleBorder` pentru a identifica ce informatie este afisata in panel-ul corespunzator.

Primele 4 panel-uri contin fiecare cate un label care are rolul de a afisa utilizatorului una dintre variabilele ce reprezinta starea actuala a simularii si sunt asezate pe un rand si 4 coloane cu ajutorul layout-ului de tip `GridBagLayout` al panel-ului in care sunt incluse.

- Aceste 4 panel-uri reprezinta, in ordine, urmatoarele variabile de stare:
- Starea curenta a simularii
- Timpul mediu de asteptare
- Timpul mediu de procesare
- Ora de varf

Penultimul panel contine ultimul panel si este extins pe 4 coloane folosind un obiect de tipul `GridBagConstraints`. Scopul lui este de a afisa log-urile generate de simulare.

Ultimul panel este un panel de tip `JScrollPane` pentru a putea oferi utilizatorului posibilitatea de a vedea totalitatea log-urilor generate de aplicatie in cazul in care acestea sunt prea mari pentru a fi afisate in fereastra. In acest panel se stocheaza log-urile prin intermediul unei componente de tip `TextArea`.

4.2.3. Fereastra de vizualizare

Fereastra de stari (**Figura 3**) este formata pe langa cele 2 panel-uri principale mentionate anterior dintr-un numar de 4 alte panel-uri: 2 perechi de `JScrollPane` si content panel. Panel-urile de tip `JScrollPane` sunt folosite pentru a oferi posibilitatea utilizatorului sa vada totalitatea elementelor afisate in cazul in care acestea nu incap in fereastra. Fiecare content panel are setata o bordura de tip `TitleBorder` pentru a identifica ce informatie este afisata in panel-ul corespunzator.

Prima pereche de panel-uri ocupa o bucata de 15% din spatiul pe orizontala al ferestrei, lucru realizat cu un set de constrangeri de tipul `GridBagConstraints`, si are rolul de a oferi o reprezentare vizuala a numarului de clienti care inca nu au ajuns sa fie asezati la coada.

A doua pereche de panel-uri ocupa o bucata de 85% din spatiul pe orizontala al ferestrei, lucru realizat cu un set de constrangeri de tipul `GridBagConstraints`, si are rolul de a oferi o reprezentare vizuala a cozilor si a clientilor care sunt asezati la coada respectiva. Aici sunt prezente pe primul rand un numar de label-uri care au o bordura de tip `TitleBorder` pentru a afisa numarul cozii si o poza cu un carucior care semnifica o coada.

Reprezentarea clinetilor se face cu niste poze in forma de omuleti negri care se vor regasi in interiorul unor label-uri.

5. Rezultate

Pentru a verifica rezultatele simularilor de cozi din cadrul aplicatiei am rulat mai multe astfel de simulari si am verificat rezultatele acestora cu rezultatele obtinute de mine pentru lista corespunzatoare de clienti generati. Verificarea se va face pe setul de valori (Timp mediu de asteptare, Timp mediu de procesare, Ora de varf).

Scenariul 1

No. clients: 10
No. servers: 3
Time limit: 5
Max load: 100
Min arrival time: 1
Max arrival time: 3
Min processing time: 1
Max processing time: 4
Selection policy: SHORTEST_TIME
Actual result: (1.40, 1.60, 3)
Expected result: (1.40, 1.60, 3)
Trecut: Da

Log-uri:

Time 0

Waiting clients: (2, 1, 1); (3, 1, 1); (1, 2, 4); (7, 2, 2); (9, 2, 4); (10, 2, 3); (4, 3, 3); (5, 3, 3); (6, 3, 4); (8, 3, 2);

Queue 1: closed

Queue 2: closed

Queue 3: closed

Time 1

Waiting clients: (1, 2, 4); (7, 2, 2); (9, 2, 4); (10, 2, 3); (4, 3, 3); (5, 3, 3); (6, 3, 4); (8, 3, 2);

Queue 1: (2, 1, 1)

Queue 2: (3, 1, 1)

Queue 3: closed

Time 2

Waiting clients: (4, 3, 3); (5, 3, 3); (6, 3, 4); (8, 3, 2);

Queue 1: (1, 2, 4)

Queue 2: (7, 2, 2), (10, 2, 3)

Queue 3: (9, 2, 4)

Time 3

Waiting clients: none

Queue 1: (1, 2, 3), (4, 3, 3), (8, 3, 2)

Queue 2: (7, 2, 1), (10, 2, 3), (6, 3, 4)

Queue 3: (9, 2, 3), (5, 3, 3)

Time 4

Waiting clients: none

Queue 1: (1, 2, 2), (4, 3, 3), (8, 3, 2)

Queue 2: (10, 2, 3), (6, 3, 4)

Queue 3: (9, 2, 2), (5, 3, 3)

Simulation terminated successfully with the following results:

Average waiting time: 1.40

Average processing time: 1.60

Peak hour: 3

Scenariul 2

No. clients: 13
No. servers: 5
Time limit: 5
Max load: 100
Min arrival time: 0
Max arrival time: 4
Min processing time: 2
Max processing time: 3
Selection policy: SHORTEST_TIME
Actual result: (2.45, 1.89, 1)
Expected result: (2.45, 1.89, 1)
Trecut: Da

Log-uri:

Time 0

Waiting clients: (2, 1, 2); (3, 1, 3); (5, 1, 2); (7, 1, 3); (11, 1, 3); (12, 1, 2); (13, 1, 2); (8, 2, 3); (4, 3, 2); (6, 4, 2); (10, 4, 3);
Queue 1: (1, 0, 3)
Queue 2: (9, 0, 2)
Queue 3: closed
Queue 4: closed
Queue 5: closed

Time 1

Waiting clients: (8, 2, 3); (4, 3, 2); (6, 4, 2); (10, 4, 3);
Queue 1: (1, 0, 2), (11, 1, 3)
Queue 2: (9, 0, 1), (7, 1, 3)
Queue 3: (2, 1, 2), (12, 1, 2)
Queue 4: (3, 1, 3)
Queue 5: (5, 1, 2), (13, 1, 2)

Time 2

Waiting clients: (4, 3, 2); (6, 4, 2); (10, 4, 3);
Queue 1: (1, 0, 1), (11, 1, 3)
Queue 2: (7, 1, 3)
Queue 3: (2, 1, 1), (12, 1, 2)
Queue 4: (3, 1, 2), (8, 2, 3)
Queue 5: (5, 1, 1), (13, 1, 2)

Time 3

Waiting clients: (6, 4, 2); (10, 4, 3);
Queue 1: (11, 1, 3)
Queue 2: (7, 1, 2), (4, 3, 2)
Queue 3: (12, 1, 2)
Queue 4: (3, 1, 1), (8, 2, 3)
Queue 5: (13, 1, 2)

Time 4

Waiting clients: none
Queue 1: (11, 1, 2)
Queue 2: (7, 1, 1), (4, 3, 2)
Queue 3: (12, 1, 1), (6, 4, 2)
Queue 4: (8, 2, 3)
Queue 5: (13, 1, 1), (10, 4, 3)

Simulation terminated successfully with the following results:

Average waiting time: 2.45

Average processing time: 1.89

Peak hour: 1

6. Concluzii

Am realizat o aplicatie Java care are capacitatea de a simula comportamentul unor clienti in cadrul unor cozi in functie de niste parametri de simulare alesi de catre utilizator prin intermediul interfetei grafice dedicate.

Acest proiect este unul pentru uz specific, se poate folosi oricand e nevoie de efectuarea unei simulari de cozi. Functioneaza fara probleme, au fost tratate diferite cazuri, astfel incat sa se obtina rezultatul dorit sau ca sa fie afisat un mesaj de eroare in cazul aparitiei unei erori.

In cadrul acestei teme am invatat sa folosesc thread-uri, metode de sincronizare intre thread-uri, sa stapanesc mai bine folosirea modelului de proiectare Strategy Pattern si sa acumulez mai multa experienta in utilizarea limbajului Java pentru a rezolva diverse probleme.

Ca dezvoltari ulterioare se pot aduce optimizari codului deja existent (modul de memorare a datelor) sau se pot implementa noi functionalitati precum posibilitatea de a folosi mai multi parametri specializati in cadrul unei simulari, un istoric al simularilor efectuate, posibilitatea de redare a unei simulari, posibilitatea de a pune pauza executiei unei simulari sau chiar posibilitatea de a importa (parametrii pentru simulare) / exporta rezultatele unor simulari (sub forma PDF, CSV, EXCEL, etc.).

7. Bibliografie

- ✓ Concurrency
 - <https://www.javacodegeeks.com/2012/12/multi-threading-in-java-swing-with-swingworker.html>
 - <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
- ✓ Executors
 - <http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html>
- ✓ Swing
 - <https://docs.oracle.com/javase/8/docs/api/java/awt>
 - <https://stackoverflow.com/>
 - <https://docs.oracle.com/javase/tutorial/uiswing/>
 - <http://asimdlv.com/java-swing-auto-scrolling-jscrollpane-i-e-chat-window/>
 - https://www.tutorialspoint.com/swingexamples/add_title_to_border_panel.htm
- ✓ Diagrame
 - <https://plantuml.com/sitemap-language-specification>
- ✓ Modelul arhitectural MVC
 - Programare Orientata Obiect (An 2, Semestrul 1)
 - <https://medium.com/@socraticsol/why-mvc-architecture-e833e28e0c76>
- ✓ Java naming conventions
 - <https://google.github.io/styleguide/javaguide.html>