DOCUMENTATIE

TEMA *2*

NUME STUDENT: Ionas Andreea-Georgiana
GRUPA: 30227

CUPRINS

1. Obiectivul temei

(i)Principalul obiectiv al acestei teme este realizarea unei aplicatii care sa simuleze o serie de N clienti care sosesc pentru servicii, intra in Q cozi, si asteapta sa fie serviti iar in cele din urma parasesc cozile.

(ii)Obiectivele secundare:

Stabilirea unor structuri de date pe care sa le folosim in proiect.	Implementam clasele “Task” si “Server” care reprezinta clientii si cozile acestei aplicatii.
Implementarea aplicatiei de simulare.	Legam toate clasele intre ele pentru ca aplicatia sa functioneze. Implementam Simulation Manager care ne va simula comportamentul cozilor.
Realizarea unei interfete grafice.	Realizam o interfata unde utilizatorul sa poata introduce datele de intrare.
Testarea aplicatiei de simulare.	Pentru niste date de intrare, date de utilizator, observam daca functionarea cozilor este corecta.

*Toate obiectivele secundare vor fi detaliate in capitolul 4.

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

Cerinte functionale:

- *Aplicatia trebuie sa simuleze comportarea unor cozi, cand se introduc cilenti si cand pleaca clienti. Se introduce un client in coada care are waiting time-ul cel mai mic (Time Strategy).*
- *Datele trebuie introduse prin intermediul unei interfete grafice.*
- *Trebuie sa se valideze datele apoi se porneste simularea cu datele primite.*
- *Se genereaza random task-uri care au arrival time si service time.*

Cerinte non functionale:

- *Aplicatia trebuie sa genereze rasunsul corect.*
- *Interfata trebuie sa fie usor de inteles si utilizat.*
- *Clientii trebuie sa fie intr-o lista de asteptare inaite de a fi pusi intr-o coada.*
- *Dupa ce serviceTime-ul unui client a trecut acesta trebuie sa fie scos din coada.*

Descriere use-case:
Use case: configurare simulare(manager cozi)
Actor principal: User-ul
Pasii pentru success:

1) *Utilizatorul introduce valorile pentru: numarul de client, numarul de cozi, intervalul de simulare, timpul minim de sosire, timpul maxim de sosire, timpul minim de servire si timpul maxim de servire.*
2) *Utilizatorul apasa butonul de "Start" pentru a valida datele de intrare.*
3) *Aplicatia valideaza datele si afiseaza un mesaj in cazul in care datele introduse au fost gresite, iar apoi incepe simularea.*

Scenariu alternativa: Valori invalide pentru parametrii de configurare

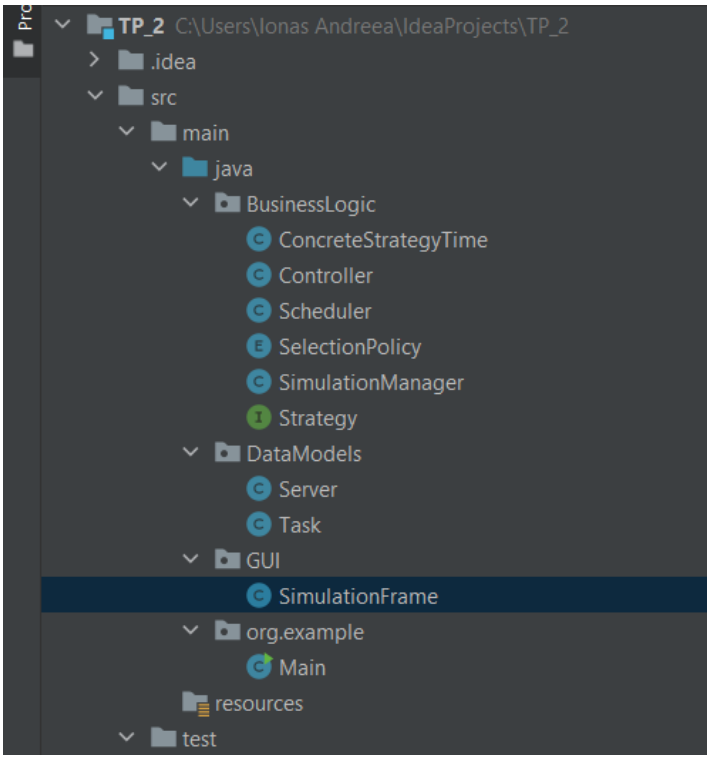
1) *Utilizatorul introduce valori invalide pentru parametrii de configurare ai aplicatiei.*
2) *Aplicatia afiseaza un mesaj de eroare si solicita utilizatorului sa introduca valori valide.*
3) *Scenariul se intoarce la pasul 1.*

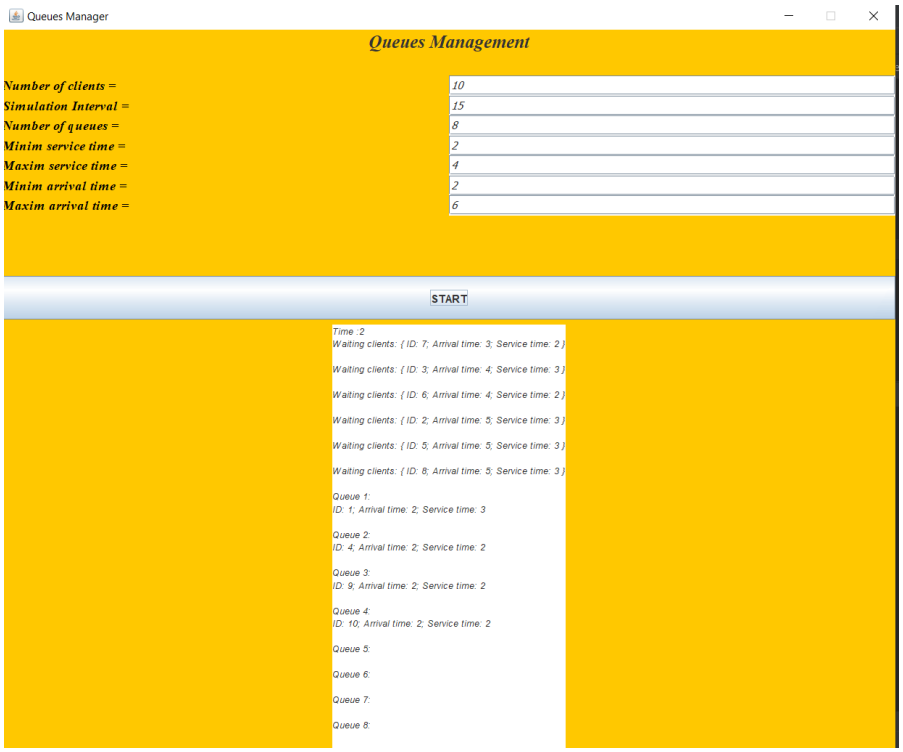
3. Proiectare

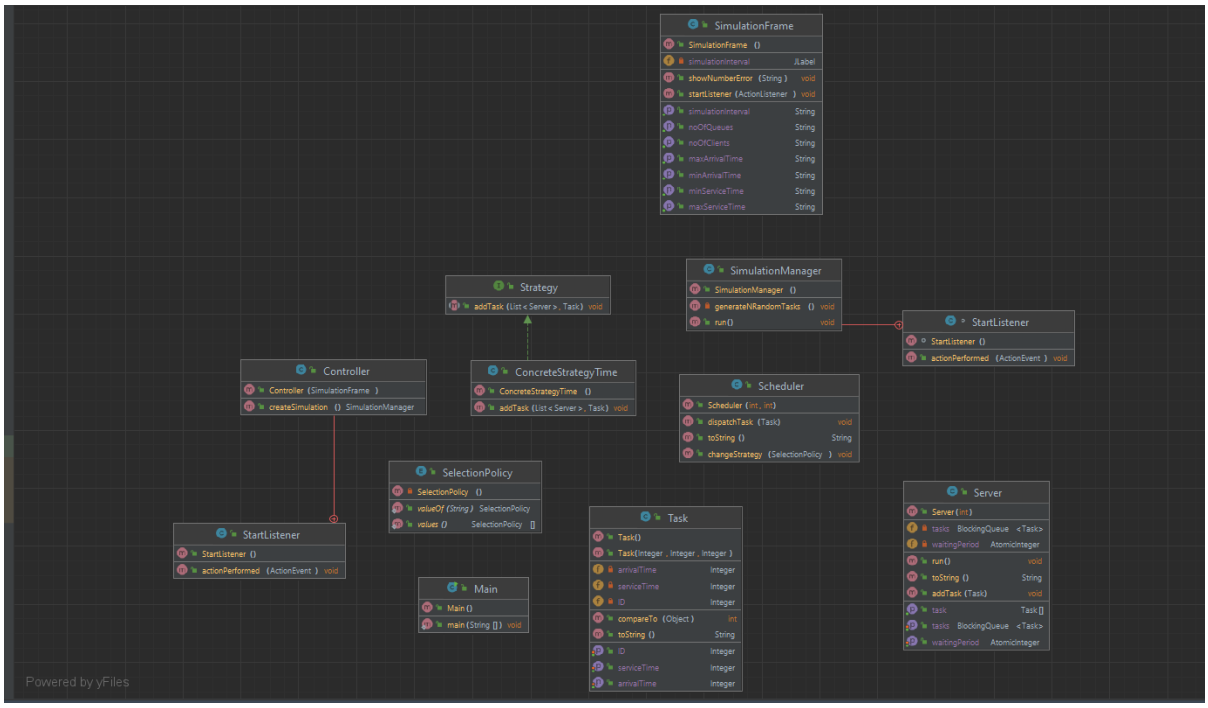
Proiectarea OOP a aplicatiei:

Divizarea proiectului in mai multe pachete si clase:

- Pachetul BuisinessLogic in care avem clasele SimulationManager, Scheduler, Strategy, ConcreteStrategyTime, SelectionPolicy si Controller.
- Pachetul DataModels in care avem clasele Server si Task.
- Pachetul GUI are clasa SimulationFrame.







Interfata cu utilizatorul este usor de inteles si utilizat. Utilizatorul trebuie sa introduca valorile pentru: maxim arrival time, maxim service time, minim arrival time, maxim service time, number of clients, number of queues, simulation interval. Valorile sunt apoi validate, dupa validare va porni simularea: clientii vor fi adaugati in cozi in functie de arrival time si vor fi scosi in functie de waiting time-ul acestora. Simularea va rula pana la valoarea simulation interval chiar daca cozile s-au golit.

4. Implementare

Clasele Task si Server sunt principalele clase ale proiectului, ele reprezinta clientii(Task) si cozile(Server). Fiecare client (Task) are ca atribute un ID, un arrival time si un service time. In clasa Task au fost implementate metode cum ar fi:

- Getters.
- Setters.
- toString().
- Constructor.

Clasa Task implementeaza Comparable, pentru a putea, mai tarziu, sa ordonam in functie de arrival time Task-urile generate. Taskurile reprezinta clientii care vor fi plasati in cozi, fiecare Task are un ID unic, un arrival time si un service time generate random. Serverele reprezinta cozile in care vor fi plasati clientii. In aceasta aplicatie vom adauga un task in coada in momentul in care arrival time-ul taskului este egal cu current time, iar task-ul va fi adaugat in coada care are waiting time-ul cel mai mic.

Fiecare coada are un BlockingQueue<Task> si un AtomicInteger care reprezinta waiting time-ul. Clasa Server implementeaza Runnable, deoarece doreste sa lucreze cu Thred-uri.

Taskurile sunt stocate intr-un Blocking Queue adica o coada care suporta operatii care sa asteapte ca coada sa devina ne-goala atunci cand se preia un element, si asteapta ca spatial sa devina disponibil in coada atunci cand se stocheaza un element.

Iar waiting time-ul este un Atomic Integer care este o clasa care ne ofera o abstarcatie atomica a unei variabile de tip intreg (int), astfel incat operatiile de citire si scriere sa fie efectuate atomic (necesitand o singura operatie). Aceasta inseamna ca nu pot exista conditii de cursa (race conditions) in care doua sau mai multe thread-uri incearca sa acceseze si sa modifice valoarea aceleasi variabile in acelasi timp. Prin urmare, “Atomic Integer” este o unealat utila pentru sincronizarea accesului concurent la o variabila intre thread-uri, cum este si cazul variabilei waiting time.

Ca metode avem:

- Getters.
- Setters.
- toString().
- Constructor.
- o metoda addTask care ne adauga un nou task in BlockingQueue.
- metoda run care este Override deoarece clasa implementeaza Runnable.
- metoda getTask care ne returneaza un task din Queue.

Clasa Scheduler care trimite task-uri in servere, are ca atribute o lista cu servere, un int maxim number of servers, un int maxim tasks per server si o strategie. Ca metode implementate avem:

- un constructor
- metoda toString().
- metoda changeStrategy care ne ofera posibilitatea de a schimba Strategya daca dorim.

- metoda dispatchTask care va adauga un task in functie de strategya aleasa.

Clasa Strategy care ne permite sa alegem politica de distribuire a clientilor. In clasa ConcreteStrategyTime se suprascrie metoda addTask in asa fel incat se alege sa se adauge task-uri in coada in care waiting time-ul este minim.

Clasa Simulation Manager are ca atribute informatiile citite din interfata grafica si pe langa un Scheduler, un Simulaton Frame si o lista de task-uri.In Simulation Manager se genereaza random task-uri(cu arrival time si service time). Aceeasta clasa este cea in care se simuleaza comportamentul cozilor in anumite conditii, cu diferite valori pentru simulation interval, number of servers, number of clients, etc.

Clasa Controller are ca attribute un Simulation Frame iar ca metode:

- Constructor.
- createSimulatinManager(), aceasta ne creaza un Simulation Manager care primeste datele introduce de utilizator.

5. Rezultate

In urma implementarii si testarii aplicatiei de queue management pot observa urmatoarele:
Pentru parametrii:

- Simulation Interval: 15
- Number of queues: 5
- Number of clients: 5
- Max Arriva Time: 6
- Max Service Time: 6
- Min Arrival Time: 2
- Min Service Time: 2

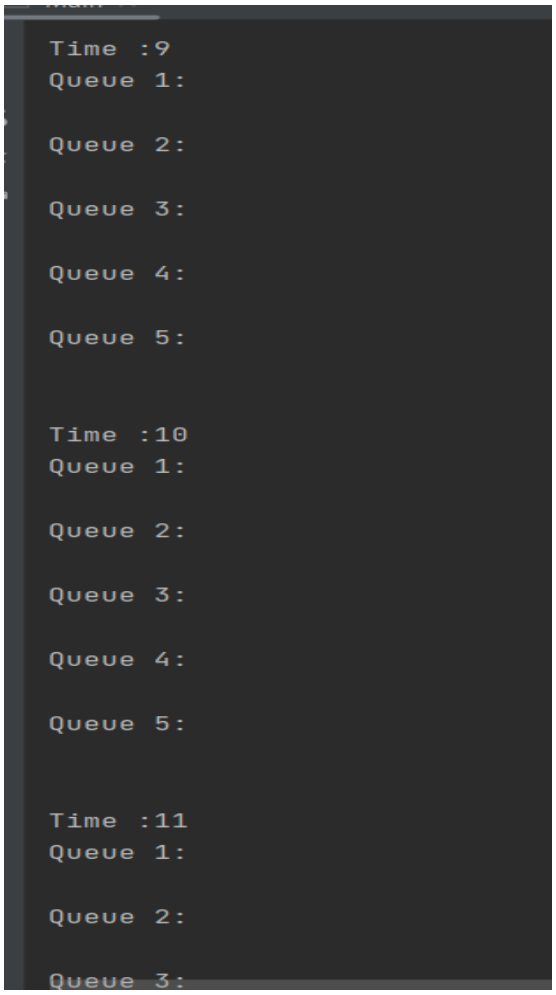
s-a generat urmatorul raspuns. Se poste observa ca la Time = 0 cozile sunt goale si avem 5 clienti in asteptare. Primul client va fi introdus la Time = 2 si va parasi coada la Time = arrival time + service time (2 + 3 = 5). Se poate observa ca la momentul de timp 5 task-ul din prima coada a fost eliminat si a fost inlocuit de altul.
La momentul Time = 9 toate cozile sunt goale, insa simularea va rula in continuare pana la Time = Simulation interval (15).

```
Time :0
Waiting clients: { ID: 1; Arrival time: 2; Service time: 3 }
Waiting clients: { ID: 5; Arrival time: 3; Service time: 2 }
Waiting clients: { ID: 2; Arrival time: 5; Service time: 4 }
Waiting clients: { ID: 3; Arrival time: 5; Service time: 3 }
Waiting clients: { ID: 4; Arrival time: 5; Service time: 3 }
Queue 1:

Queue 2:

Queue 3:

Queue 4:

Queue 5:


Time :1
Waiting clients: { ID: 1; Arrival time: 2; Service time: 3 }
Waiting clients: { ID: 5; Arrival time: 3; Service time: 2 }
Waiting clients: { ID: 2; Arrival time: 5; Service time: 4 }
Waiting clients: { ID: 3; Arrival time: 5; Service time: 3 }
Waiting clients: { ID: 4; Arrival time: 5; Service time: 3 }
Queue 1:

Queue 2:

Queue 3:

Queue 4:
```

```
Time :2
Waiting clients: { ID: 5; Arrival time: 3; Service time: 2 }
Waiting clients: { ID: 2; Arrival time: 5; Service time: 4 }
Waiting clients: { ID: 3; Arrival time: 5; Service time: 3 }
Waiting clients: { ID: 4; Arrival time: 5; Service time: 3 }
Queue 1:
ID: 1; Arrival time: 2; Service time: 3

Queue 2:

Queue 3:

Queue 4:

Queue 5:


Time :3
Waiting clients: { ID: 2; Arrival time: 5; Service time: 4 }
Waiting clients: { ID: 3; Arrival time: 5; Service time: 3 }
Waiting clients: { ID: 4; Arrival time: 5; Service time: 3 }
Queue 1:
ID: 1; Arrival time: 2; Service time: 3

Queue 2:
ID: 5; Arrival time: 3; Service time: 2

Queue 3:
```

```
Time :4
Waiting clients: { ID: 2; Arrival time: 5; Service time: 4 }
Waiting clients: { ID: 3; Arrival time: 5; Service time: 3 }
Waiting clients: { ID: 4; Arrival time: 5; Service time: 3 }
Queue 1:
ID: 1; Arrival time: 2; Service time: 3

Queue 2:
ID: 5; Arrival time: 3; Service time: 2

Queue 3:

Queue 4:

Queue 5:


Time :5
Queue 1:
ID: 3; Arrival time: 5; Service time: 3

Queue 2:
ID: 4; Arrival time: 5; Service time: 3

Queue 3:
ID: 2; Arrival time: 5; Service time: 4

Queue 4:

Queue 5:
```

```
Time :6
Queue 1:
ID: 3; Arrival time: 5; Service time: 3

Queue 2:
ID: 4; Arrival time: 5; Service time: 3

Queue 3:
ID: 2; Arrival time: 5; Service time: 4

Queue 4:

Queue 5:


Time :7
Queue 1:
ID: 3; Arrival time: 5; Service time: 3

Queue 2:
ID: 4; Arrival time: 5; Service time: 3

Queue 3:
ID: 2; Arrival time: 5; Service time: 4

Queue 4:

Queue 5:
```



Pentru cele 3 cazuri de date primite la laborator s-a rulat aplicatia si s-au generat 3 log-uri (log1, log2, log3).

Pentru log1 datele primate au fost:

- Tasks = 4
- Servers = 2
- Simulation Time = 60
- Min Arriva Time = 2
- Max Arrival Time = 30
- Min Service Time = 2
- Max Service Time = 4

Pentru log2 datele primate au fost:

- Tasks = 50
- Servers = 5
- Simulation Time = 60

- Min Arriva Time = 2
- Max Arrival Time = 40
- Min Service Time = 1
- Max Service Time = 7

Pentru log3 datele primate au fost:

- Tasks = 1000
- Servers = 20
- Simulation Time = 200
- Min Arriva Time = 10
- Max Arrival Time = 100
- Min Service Time = 3
- Max Service Time = 9

6. Concluzii

In timpul studierii temei acesteia am invatat ca un thread este o cale de a perimte programelor sa execute mai multe sarcini in paralel. Am invatat cum sa creez si sa controlez thread-urile, cum sa sincorinizez accesul la resurse impartite intre thread-uri si cum sa tratez problemele de concurenta.

De asemenea, am invatat despre diversele clase si interfete Java care sunt utile pentru lucrul cu thread-uri, cum ar fi clasa Thread, interfata Runnable, metoda run () si Atomic Integer.

Un aspect care ar putea fi imbunatatit la aceasta tema este aplicarea unor concept mai avansate de concurenta, cum ar fi deadlock si livelock, si dezvoltarea de strategii mai eficiente de sincronizare a thread-urilor.

De asemenea, as putea sa explorez biblioteci si framework-uri externe care ofera functionalitati avansate pentru lucruri cu thread-uri, cum ar fi Executor Framework si Akka.

In viitor la acest proiect s-ar putea creea strategi noi de sortare a cozilor, s-ar putea imbunatatii interfata cu utilizatorul si s-ar putea optimiza timpul de procesare si executare.

7. Bibliografie

- *Java Threads - https://www.w3schools.com/java/java_threads.asp*
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html
- *Java Files – https://docs.oracle.com/javase/7/docs/api/java/io/File.html*
  - https://www.programiz.com/java-programming/file
- *Atomic Integer - https://www.tutorialspoint.com/java_concurrency/concurrency_atomic_integer.htm*

- *Blocking Queue - https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html*