DOCUMENTATIE

TEMA *2*

NUME STUDENT: Buzila Maria-Alexandra
GRUPA: 30226

CUPRINS

1. Obiectivul temei

Obiectiv principal al acestei teme este proiectarea și implementarea unei aplicații care are ca scop analiza sistemelor bazate pe cozi de așteptare prin simularea unei serii de N clienți care sosesc pentru service, intră în cozile Q, așteaptă, sunt serviți și în cele din urmă părăsesc cozile și prin calcularea timpului mediu de așteptare, a timpului mediu de serviciu și a orelor de vârf.

Obiectivele secundare ce ajuta la realizarea celui principal sunt prezentate succint in lista de mai jos.

• Analizarea problemei și identificarea cerințelor (obiectiv descris in capitolul 2)

- Se determina cerintele functionale si cele non-functionale

• Proiectarea aplicației de simulare (obiectiv descris in capitolul 3)

- Utilizarea resurselor pe care le avem la dispozitie pentru a realiza legaturile importante, care ne vor ajuta cel mai mult la realizarea propriu-zisa a acesteia
- Realizarea unui plan/sistem eficient, corect si cat mai usor de implementat: determinam clasele, legaturile dintre ele si organizarea lor in pachete

• Implementarea aplicației de simulare (obiectiv descris in capitolul 4)

- Aceasta etapa este reprezentata de scrierea de cod in limbaj Java a claselor corespunzatoare, impreuna cu metodele acestora.

• Testați aplicația de simulare (obiectiv descris in capitolul 5)

- Testarea este cea mai importanta etapa in realizarea unei astfel de aplicatie
- Testam aplicatie folosind testele date in cerintele acestei teme

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

In continuare, voi explica cateva din cerintele functionale (ce descriu ce trebuie sa faca programul) si non-functionale (ce descriu calitatile pe care ar trebui sa le aiba programul) ce trebuie bine intelese.

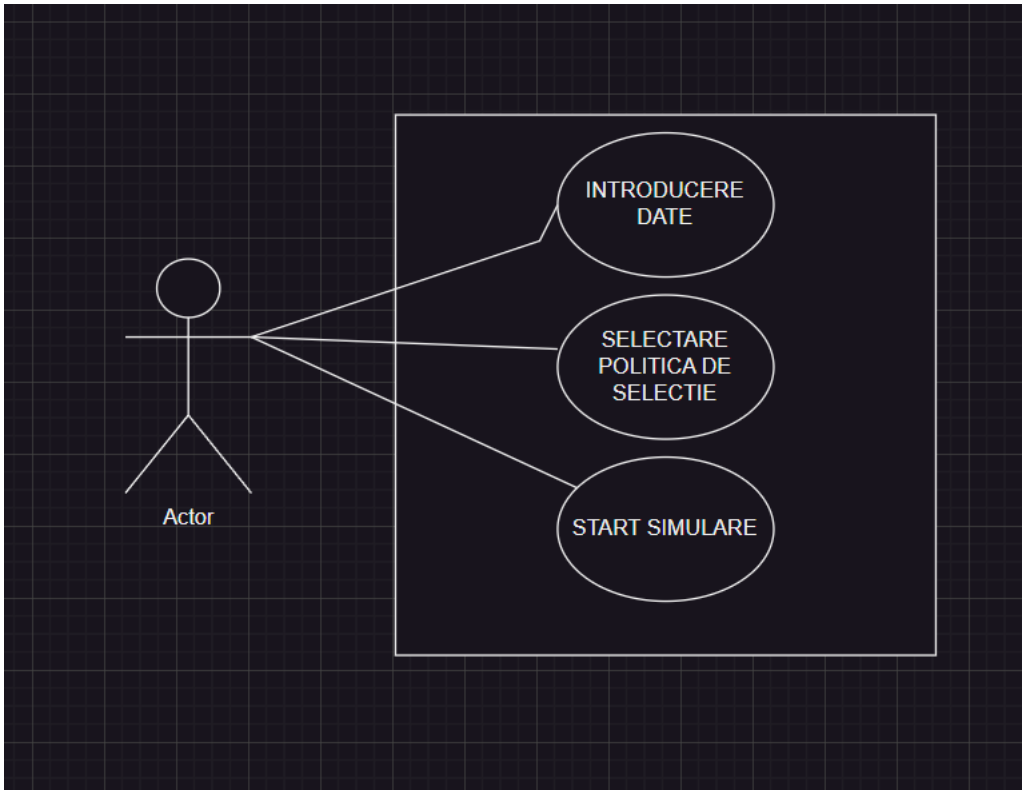
Cerinte functionale:

- Aplicația de simulare ar trebui să permită utilizatorilor să configureze simularea
- Aplicația de simulare ar trebui să permită utilizatorilor să înceapă simularea
- Aplicația de simulare ar trebui să afișeze evolutia cozilor în timp real

Cerinte non-functionale:

- Aplicația de simulare ar trebui să fie intuitivă și ușor de utilizat utilizare de către utilizator
- Aplicatia trebuie sa precizeze clar datele de intrare ce trebuie introduse, iar mediul de preluarea a datelor sa fie usor de folosit

Diagrama use-case:



Use Case: Initializare simulare
Actor principal: utilizator
Principalul scenariu de succes:

1. Utilizatorul introduce valorile pentru: numărul de clienți, numărul de cozi, intervalul de simulare, minimul și maximul timpului de sosire și minimul și maximul timpului de servire
2. Utilizatorul alege politica de selectie
3. Utilizatorul apasa pe buntonul “START”
4. Aplicația validează datele și afișează un mesaj informând utilizatorul că începe simularea

Secvență alternativă: valori nevalide pentru parametrii de configurare

- Utilizatorul introduce valori invalide pentru configurarea aplicației
- Aplicația afișează un mesaj de eroare utilizatorului
- Scenariul revine la pasul 1

3. Proiectare

Design – Conceptual Architecture

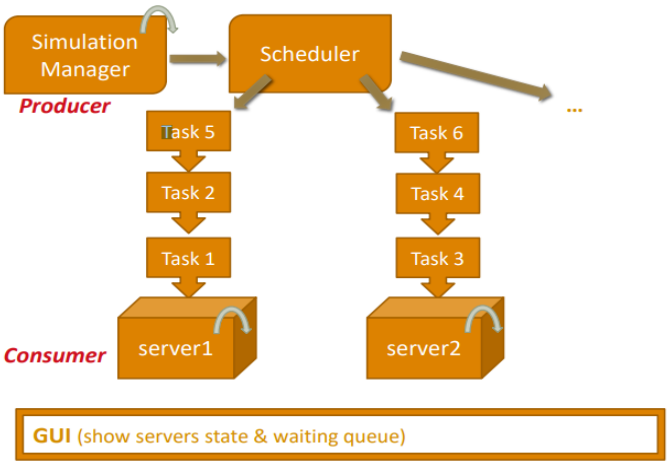
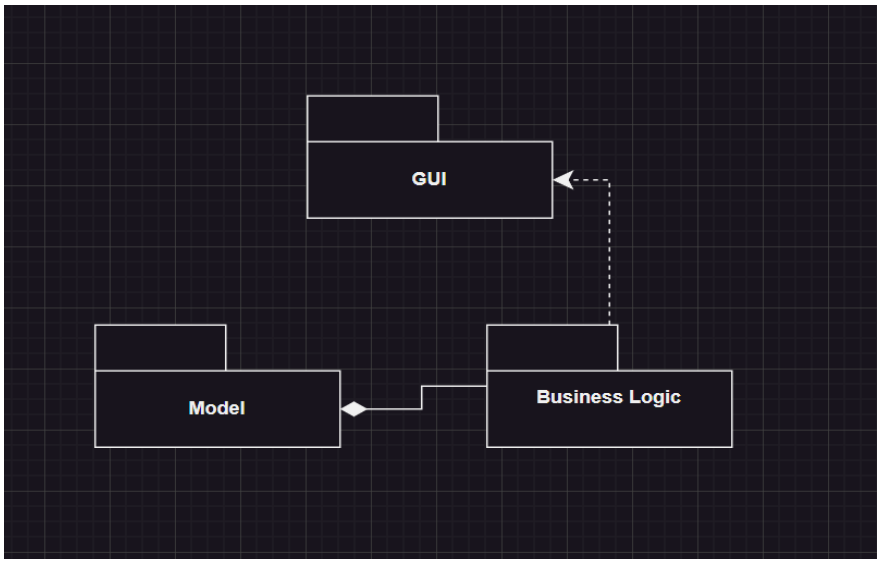
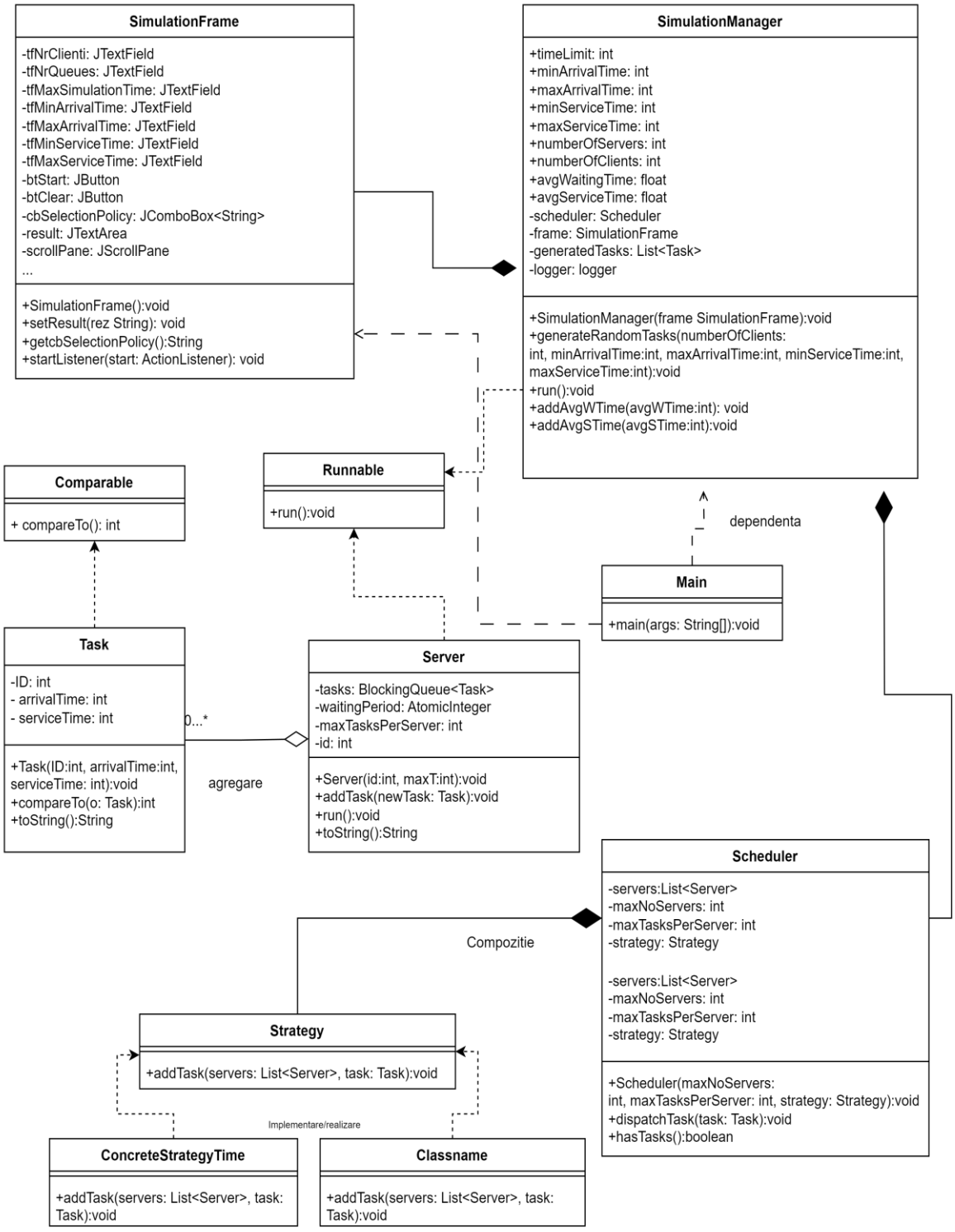


Diagrama de pachete:



(dependenta, compozitie)

Diagrama de clase:



- In implementare am folosit ca si structuri de date BlockingQueue<Task>, implementata de ArrayBlockingQueue<Task>. De asemenea, am folosit variabile atomice de tipul AtomicInteger si o enumerare pentru tipul strategiei care are doua variante: SHORTEST_QUEUE, SHORTEST_TIME.
- Iar interfetele implementate pot fi observate in diagrama de clase

4. Implementare

1. Clasa Task

Atribute: “ID” care reprezinta numarul unic al fiecarui client/task, “arrivalTime” -> timpul la care task-ul ajunge in coada si “serviceTime” -> timpul petrecut in coada

Metode: Avem gettere, toString si deoarece implementam interfata Comparable avem si functia compareTo care ajuta la sortarea in ordine crescatoare a task-urile in functie de timpul de sosire;

```
@Override
public int compareTo(Task o) {
    return this.arrivalTime - o.arrivalTime;
}
```

2. Clasa Server

Atribute: Contine un “id” ce ajuta la identificarea acestuia, o coada de task-uri si timpul de asteptare pentru coada.

Metode: Deoarece implementeaza interfata Runnable, avem metoda run() care realizeaza “servirea” clientilor, adica decrementeaza timpul de servire al acestora dupa fiecare perioada de tim, iar atunci cand acesta ajunge la 0, clientul este scos din coada.

```
@Override
public void run() {
    while(true) {
        while (tasks.peek() != null) {
            Task t = tasks.peek();
            try {
                Thread.sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            this.waitingPeriod.getAndDecrement();
            t.setServiceTime(t.getServiceTime() - 1);
            if (t.getServiceTime() == 0) {
                try {
                    tasks.take();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

3. **Clasa ConcreteStrategyQueue**

Metode: implementeaza interfata Strategy => avem metoda addTask – care adauga urmatorul client in cea mai scurta coada

```java
@Override
public void addTask(List<Server> servers, Task task) {
    Server minServer = servers.getFirst();
    for(Server server : servers){
        if(server.getTasks().size() < minServer.getTasks().size() && server.getTasks().size() < server.getMaxTasksPerServer()){
            minServer = server;
        }
    }
    SimulationManager.addAvgWTime(minServer.getWaitingTime());
    minServer.addTask(task);
}
```

4. **Clasa ConcreteStrategyTime**

Metode: implementeaza interfata Strategy => avem metoda addTask – care adauga urmatorul client in coada care are timpul de asteptare cel mai mic

5. **Clasa Scheduler**

Atribute: Contine o lista cu servere, numarul de servere, numarul maxim de task-uri dintr-un server si strategia conform careia se face adaugarea unui task la server.

- Aceasta creeaza serverele si porneste un thread pentru fiecare in parte;

Metode: “dispatchTask” care adauga task-urile in cozi in functie de strategia aleasa de utilizator si “hasTasks” care e o metode de verificare (daca mai exista sau nu task-uri in servere);

6. **Clasa Simulation Manager**

Atribute: toate datele de intrare, variabile float pentru calcularea timpului mediu de servire si asteptare, o lista pentru generarea random a task-urilor si un “logger” pentru realizarea fisierelor cu rezultate

- Aceasta ia datele din SimulationFrame si le filtreaza; de asemenea deschide fisierul de scriere cu formatul corespunzator

Metode: pentru generarea random a task-urilor in functie de datele date de utilizator si pentru calcularea timpilor medii, functii statice

Pentru ca implementeaza interfata Runnable, avem metoda run() , in care scriem atat in fisier cat si prezentam utilizatorului evalotia aplicatiei in timp real; pentru fiecare perioada de timp din intervalul de simulare se adauga clienti in cozile corespunzatoare in functie de strategia aleasa de

user. Pentru a putea prelua datele modificate adormim thread-ul 1 secunda. Tot aici calculam si peak_hour.

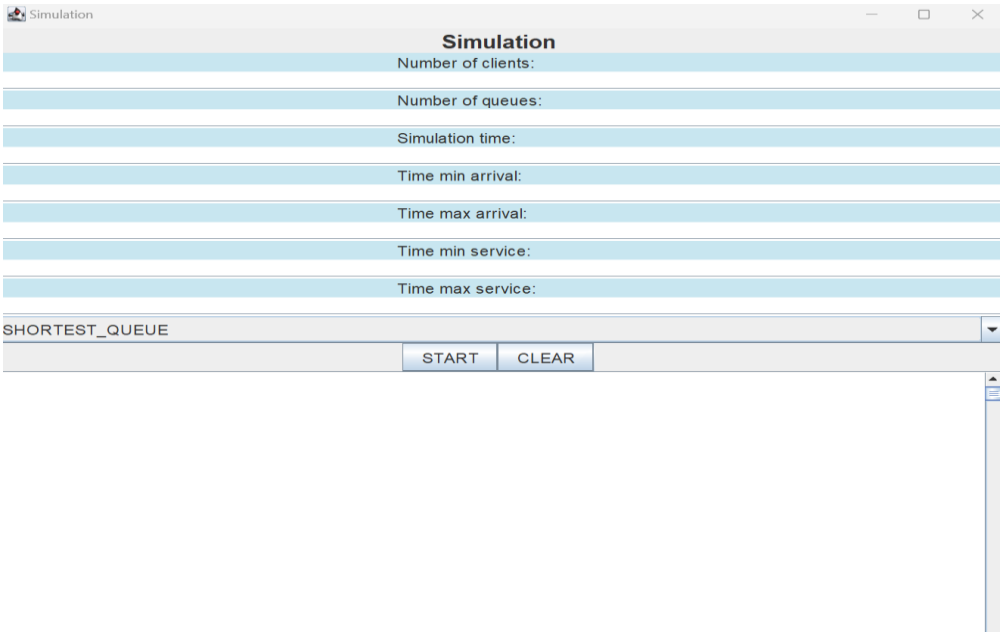
```
@Override
public void run() {
    String simulation = "Simulation started\n";
    int peak_hour = 0;
    int maxTasks = 0;
    frame.SetResult(simulation);
    int currentTime = 0;
    while(currentTime < timeLimit) {
        int aux = 0;
        while (!generatedTasks.isEmpty() && generatedTasks.get(0).getArrivalTime() == currentTime) {
            Task t = generatedTasks.get(0);
            if (t.getArrivalTime() + t.getServiceTime() < timeLimit) {
                scheduler.dispatchTask(t);
                addAvgSTime(t.getServiceTime());
                generatedTasks.remove( index: 0);
            }
        }
        simulation += "\nTime: " + currentTime + "\n";
```

```
simulation += "\nTime: " + currentTime + "\n";
frame.SetResult("Time: " + currentTime + "\n");
simulation += "Waiting clients: " + generatedTasks + "\n";
frame.SetResult("Waiting clients: " + generatedTasks + "\n");
for (Server s : scheduler.getServers()) {
    aux += s.getTasks().size();
    simulation += s.toString() + "\n";
    frame.SetResult(s.toString() + "\n");
}
if (aux > maxTasks) {
    maxTasks = aux;
    peak_hour = currentTime;
}
currentTime++;
try {
    Thread.sleep( millis: 1000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

```
    if(generatedTasks.isEmpty() && !scheduler.hasTasks()){
        break;
    }
}
    simulation += "\nTime: " + currentTime + "\n";
    frame.SetResult("Time: " + currentTime + "\n");
    simulation += "Waiting clients: " + generatedTasks + "\n";
    frame.SetResult("Waiting clients: " + generatedTasks + "\n");
    for (Server s : scheduler.getServers()) {
        simulation += s.toString() + "\n";
        frame.SetResult(s.toString() + "\n");
    }
    simulation += "\n" + "Simulation ended\n";
    frame.SetResult("Simulation ended\n");
    float avgWT = avgWaitingTime / (float)numberOfClients;
    float avgST = avgServiceTime / (float)numberOfClients;
    frame.SetResult("Average waiting time: " + avgWT + "\n");
    simulation += "Average waiting time: " + avgWT + "\n";
    frame.SetResult("Average service time: " + avgST + "\n");
    simulation += "Average service time: " + avgST + "\n";
    frame.SetResult("Peak hour: " + peak_hour + "\n");
    simulation += "Peak hour: " + peak_hour + "\n";
```

7. **Clasa SimulationFrame**

Atribute: contine toate elementele utilizate pentru a realiza interfata cu Swing
Metode: avem constructorul in care initializam si organizam elementele; avem un ascultator pentru butonul de CLEAR care sterge toate casetele interfetei; de asemenea, o functie ce doar adauga un asculator pentru butonul START (ascultator ce va fi pornit doar in clasa Main atunci cand sunt initializate si SimulationFrame si SimulationManager)



5. Rezultate

Test 1	Test 2	Test 3
N = 4 Q = 2 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 30]$ $[t_{service}^{MIN}, t_{service}^{MAX}] = [2, 4]$	N = 50 Q = 5 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 40]$ $[t_{service}^{MIN}, t_{service}^{MAX}] = [1, 7]$	N = 1000 Q = 20 $t_{simulation}^{MAX}$ = 200 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [10, 100]$ $[t_{service}^{MIN}, t_{service}^{MAX}] = [3, 9]$

Test 1:

Simulation started
Time: 0
Waiting clients: [(0, 4, 3), (1, 8, 2), (2, 9, 2), (3, 13, 2)]
Queue 0:closed
Queue 1:closed
Time: 1
Waiting clients: [(0, 4, 3), (1, 8, 2), (2, 9, 2), (3, 13, 2)]
Queue 0:closed
Queue 1:closed

.....

Time: 8
Waiting clients: [(2, 9, 2), (3, 13, 2)]
Queue 0:(1, 8, 2);
Queue 1:closed

Time: 9
Waiting clients: [(3, 13, 2)]
Queue 0:(1, 8, 1);
Queue 1:(2, 9, 2);

Time: 10
Waiting clients: [(3, 13, 2)]
Queue 0:closed
Queue 1:(2, 9, 1);

Time: 11
Waiting clients: [(3, 13, 2)]
Queue 0:closed
Queue 1:closed

…

Time: 15
Waiting clients: []
Queue 0:closed
Queue 1:closed

Simulation ended
Average waiting time: 0.0
Average service time: 2.25
Peak hour: 9

Test 2:

Simulation started

Time: 0
Waiting clients: [(15, 2, 2), (18, 2, 4), (45, 2, 4), (39, 3, 5), (42, 3, 2), (34, 6, 6), (47, 6, 6), (14, 7, 2), (20, 7, 5), (40, 7, 4), (30, 8, 2), (0, 9, 2), (31, 9, 2), (10, 10, 3), (41, 10, 6), (8, 11, 2), (29, 11, 3), (25, 12, 6), (7, 16, 6), (5, 17, 4), (9, 17, 5), (49, 17, 4), (12, 18, 3), (13, 18, 1), (16, 18, 1), (48, 18, 3), (23, 22, 1), (2, 23, 1), (38, 23, 1), (36, 24, 1), (46, 26, 2), (35, 27, 6), (37, 27, 4), (4, 28, 2), (22, 30, 4), (11, 31, 1), (1, 32, 5), (21, 32, 5), (26, 32, 1), (28, 32, 1), (17, 35, 1), (19, 35, 1), (44, 35, 5), (3, 36, 2), (32, 37, 4), (33, 37, 6), (6, 38, 6), (27, 38, 5), (43, 38, 1), (24, 39, 5)]
Queue 0:closed
Queue 1:closed
Queue 2:closed
Queue 3:closed
Queue 4:closed

….

Time: 32
Waiting clients: [(17, 35, 1), (19, 35, 1), (44, 35, 5), (3, 36, 2), (32, 37, 4), (33, 37, 6), (6, 38, 6), (27, 38, 5), (43, 38, 1), (24, 39, 5)]
Queue 0:(22, 30, 2); (28, 32, 1);
Queue 1:(35, 27, 1);
Queue 2:(1, 32, 5);
Queue 3:(21, 32, 5);
Queue 4:(26, 32, 1);

Time: 33

Waiting clients: [(17, 35, 1), (19, 35, 1), (44, 35, 5), (3, 36, 2), (32, 37, 4), (33, 37, 6), (6, 38, 6), (27, 38, 5), (43, 38, 1), (24, 39, 5)]
Queue 0:(22, 30, 1); (28, 32, 1);
Queue 1:closed
Queue 2:(1, 32, 4);
Queue 3:(21, 32, 4);
Queue 4:closed

…

Simulation ended
Average waiting time: 0.82
Average service time: 3.28
Peak hour: 11

Test 3:

….
Time: 199
Waiting clients: []
Queue 0:(411, 98, 4); (885, 99, 3);
Queue 1:(206, 97, 7); (493, 98, 8); (913, 99, 4);
Queue 2:(464, 97, 3);
Queue 3:(491, 97, 1); (125, 99, 6);
Queue 4:closed
Queue 5:(596, 98, 2);
Queue 6:(182, 97, 2); (637, 97, 8);
Queue 7:(657, 97, 6); (598, 98, 7);
Queue 8:(616, 91, 4); (158, 94, 7); (714, 94, 3); (897, 97, 7);
Queue 9:(801, 94, 1); (183, 97, 7); (717, 98, 6);
Queue 10:(764, 98, 3);
Queue 11:closed
Queue 12:(157, 98, 5); (804, 98, 6);
Queue 13:closed
Queue 14:(233, 95, 3); (774, 95, 8); (196, 98, 4); (862, 98, 3);
Queue 15:(973, 98, 3);
Queue 16:closed
Queue 17:(406, 99, 1);
Queue 18:(289, 96, 2); (273, 98, 3); (421, 99, 6);
Queue 19:closed

Time: 200

Waiting clients: []
Queue 0:(411, 98, 2); (885, 99, 3);
Queue 1:(206, 97, 6); (493, 98, 8); (913, 99, 4);
Queue 2:(464, 97, 2);
Queue 3:(125, 99, 6);
Queue 4:closed
Queue 5:closed
Queue 6:(637, 97, 8);
Queue 7:(657, 97, 4); (598, 98, 7);
Queue 8:(616, 91, 2); (158, 94, 7); (714, 94, 3); (897, 97, 7);
Queue 9:(183, 97, 6); (717, 98, 6);
Queue 10:(764, 98, 1);
Queue 11:closed
Queue 12:(157, 98, 3); (804, 98, 6);
Queue 13:closed
Queue 14:(233, 95, 1); (774, 95, 8); (196, 98, 4); (862, 98, 3);
Queue 15:(973, 98, 1);
Queue 16:closed
Queue 17:closed
Queue 18:(273, 98, 3); (421, 99, 6);
Queue 19:closed

Simulation ended
Average waiting time: 79.884
Average service time: 5.541
Peak hour: 99

- Fisierele rezultate complete sunt atasate temei.

6. Concluzii

Din aceasta tema am invatat sa lucrez mult mai bine cu thread-uri si cu cozi. De asemenea, am descoperit interfata Runnable si chiar am implementat o interfata “Strategy”. Imbunatatiri pot fi la organizarea in interfata si aspectul acesteia, cat si mult mai multe optiuni de introdus date (+se putea introduce si numarul maxim de task-uri ce se pot pune intr-o coada).

7. Bibliografie

1. *https://dsrl.eu/courses/pt/*
2. *https://mariaiulianadascalu.files.wordpress.com/2012/10/claseobiecte.pdf*
3. *https://www.w3schools.com/java/java_threads.asp*
4. *https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html*