Tehnici de programare

Tema 2 – documentație

Profesor îndrumător: David Șera

Student: Astalîș Lorena-Maria

Grupa: 30224

An de studiu: 2

2021

1. Obiectivul temei

Proiectarea și implementarea unei aplicații de simulare care vizează analizarea sistemelor de determinare și minimizare a timpului de așteptare al clienților. Cozile sunt folosite frecvent pentru a modela domenii din lumea reală. Obiectivul principal este de a plasa un ”client” să aștepte pentru a primi un ”serviciu”. Un mod de a minimiza timpul de așteptare este de a adăuga mai multe servere, mai multe cozi sistemului (fiecare coadă este presuspus că are un procesor asociat) dar în acest caz ar mări mult costul celui care oferă serviciile.

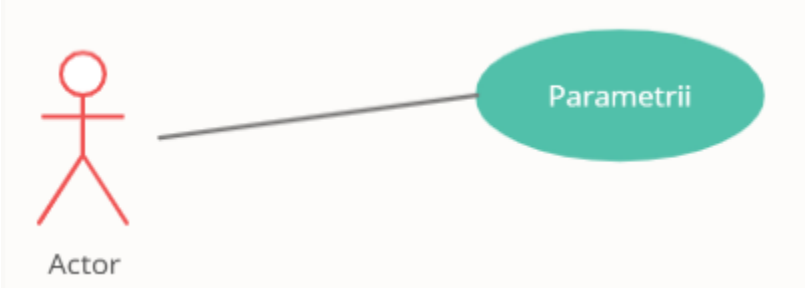
Aplicația ar trebui să simuleze (într-un timp de simulare definit $t_{simulation}$) o serie de N clienți care ajung pentru un serviciu, intrând în Q cozi, așteptând să fie serviți, iar în final să părăsească coada. Toți clienții sunt generați când simularea începe și sunt caracterizați de: ID, timpul la care a ajuns ($t_{arrival}$) și timpul de procesare ($t_{processing}$). Aplicația urmărește timpul total de așteptare a fiecărui client în coadă și calculează timpul total de așteptare.

Următoarele date ar trebui să fie considerate ca fiind date de intrare pentru aplicație care vor fi introduse de către utilizator în interfața grafică:

- Numărul clienților: N;
- Numărul cozilor: Q;
- Durata simulării: $t_{simulation}$;
- Intervalul de timp pentru sosirea clienților: ${t_{arrival}}^{MIN} < t_{arrival} < {t_{arrival}}^{MAX}$;
- Intervalul de timp pentru durata servirii unui client: ${t_{processing}}^{MIN} < t_{processing} < {t_{processing}}^{MAX}$;

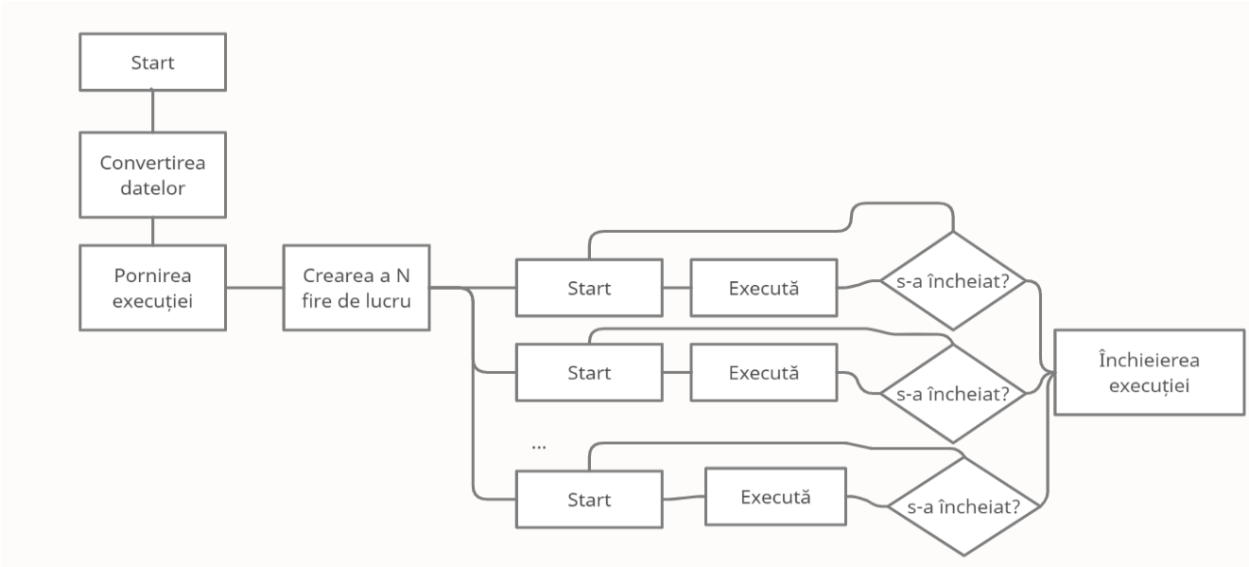
2. Analiza problemei

În primul rând interacțiunea cu utilizatorul este una redusă din cauza faptului că această aplicație este un simulator. Singurele date primite de la acesta sunt parametrii menționați mai sus în cerință.



Desigur aici putem primi date invalide de la utilizator. Acestea sunt semnalate prin excepții.

În continuare va trebui să modelăm un algoritm schițat pentru a pune cap la cap cerințele și pentru a avea un punct de plecare în implementare.



Așadar, la un mod foarte minimalist avem trasată mai sus organigrama simulatorului. După cum se poate observa conține partea inițială, în care preluăm parametrii de care avem nevoie. Convertirea datelor, aici ar intra și generarea a N clienți, fiecare cu un identificator unic, precum și un timp de sosire și un timp de procesare, aceștia fiind generați în mod aleatoriu din intervalul specificat. Pornirea execuției înseamnă crearea unui fir de execuție principal. Din acest fir principal se vor naște alte N fire de lucru. Clienții generați anterior se vor ordona după timpul de sosire. De menționat ar fi că în firul de lucru principal ținem cronometrul pentru întreaga simulare. Pe măsură ce înaintăm în execuție, adică trece o secundă, acesta se incrementează cu 1. La fiecare moment se verifică toată lista clienților, dacă timpul de sosire este egal cu momentul curent, atunci aceștia vor fi preluați de pe această listă și asignați unei cozi. Există 2 posibilități de a adăuga clienți în cozi, și anume. Punem clienți în coada cu timpul total de procesare cel mai mic, sau unde numărul de clienți din coadă este cel mai mic. De specificat este faptul că pentru testele finale am ales prima variantă deoarece este mai eficientă.

Aceasta este idea de rezolvare, pe larg a acestei teme. În capitolul următor se va face analiza mai detaliată a structurii pe care o va avea aceasta.

3. Proiectare - decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, algoritmi

S-a pornit de la ideile de bază și anume: clientul și coada, două clase esențiale.

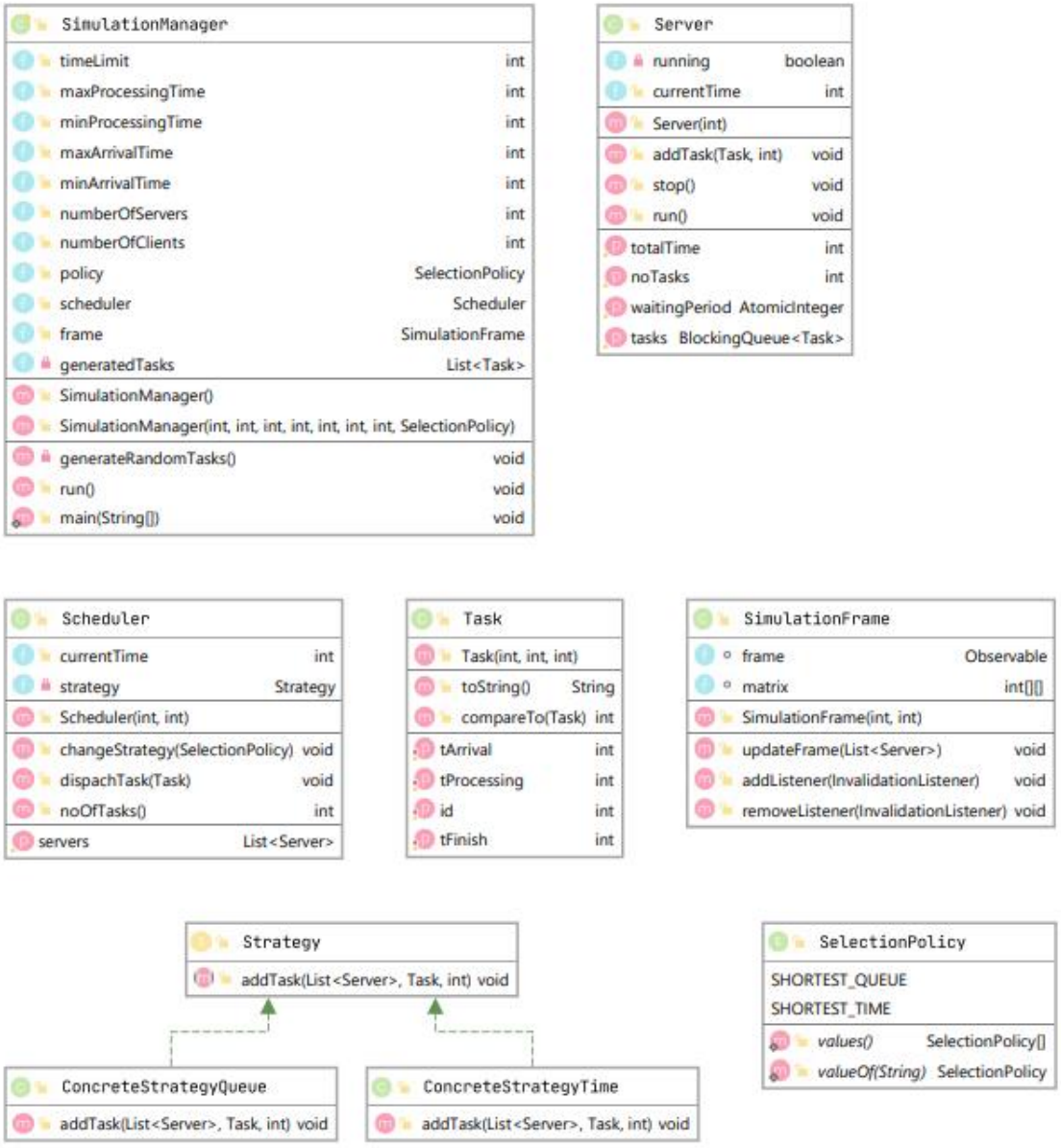
Clasa client este denumită în proiect ca și clasa Task, iar clasa coadă cu numele de Server.

Clientul are 3 proprietăți care îl definesc: identificatorul unic, timpul la care a ajuns și timpul de procesare.

Serverul este format la rândul lui dintr-o listă de Task-uri.

De asemenea avem nevoie de o strategie de a pune clienții în cozi. Aceasta este clasa Strategy, vom vedea ulterior în capitolul 4.

Clasa care este responsabilă să facă programările clienților la cozi este Scheduler. Care primește o strategie și în funcție de aceasta va adăuga Task-uri la Servere.



După cum se poate observa, aici este prezentată diagrama UML a claselor pentru temă.

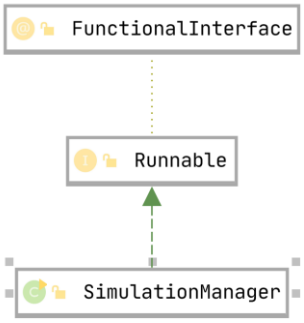
Există 2 strategii pe care le putem folosi pentru a pune clienți în coadă. Prima dintre ele ar fi plasarea clientului în coada cu timp minim. A doua este de a găsi coada cu cel mai mic număr de clienți și plasarea noului client în aceasta.

Pentru a pune toate aceste aspecte discutate cap la cap vom avea nevoie de un Planificator (Scheduler) aici ținem timpul client strategia aleasă. Aceasta are 3 metode: schimbarea strategiei, plasarea task-ului la serverul potrivit și o metodă care returnează numărul de task-uri curente.

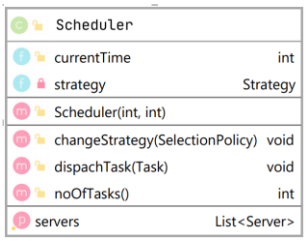
Cel mai înalt nivel de la care se poate privi modelul este clasa SimulationManager (Managerul Simulării). Aici se leagă toate clasele între ele, aici plasăm parametrii de la utilizator, generarea în mod aleator a clienților în funcție de datele primite.

4. Implemenatare

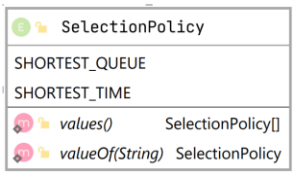
Întru-cât această aplicație funcționează pe fire de lucru, fiecare server are un thread care așteaptă să primească task-uri , a-i procesa și a trece la următorii.



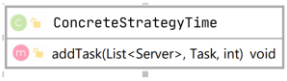
Clasa de top, SimulationManager implementează interfața Runnable. Această interfață cere a-i fi implementată metoda run().



Scheduler, parte a clasei SimulationManager, Planificatorul, precum este explicat în capitolul anterior, se ocupă cu a plasa task-urile în server.



Un enum care conține tipurile de strategii pe care le poate alege pentru a plasa clienții în cozi.



Aceste 2 clase sunt implementările concrete a strategiilor de plasare a clienților în cozi.

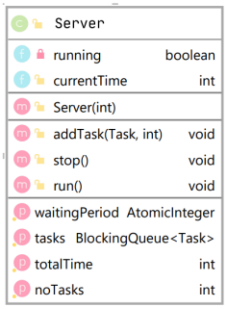
```
public class ConcreteStrategyTime implements Strategy {
    @Override
    public void addTask(List<Server> servers, Task t, int currentTime) {
        Server minimum = servers.get(0);
        for(Server server: servers) {
            if(server.getTotalTime() < minimum.getTotalTime()) {
```

```
                server.currentTime = currentTime;
                minimum = server;
            }
        }
        System.out.println("added task " + t.gettArrival() + " at server " +
minimum.toString());
        minimum.addTask(t, currentTime);
    }
}
```

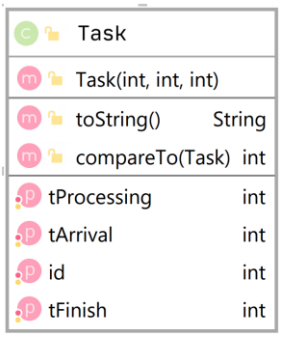
Implementarea propriu zisă a clientului în coadă pentru timpul minim.

```
public class ConcreteStrategyQueue implements Strategy {
    @Override
    public void addTask(List<Server> servers, Task t, int currentTime) {
        Server minimum = servers.get(0);
        for(Server server: servers) {
            if(server.getNoTasks() < minimum.getNoTasks()) {
                server.currentTime = currentTime;
                minimum = server;
            }
        }
        System.out.println("added task " + t.gettProcessing() + " at server " +
minimum.toString());
        minimum.addTask(t, currentTime);
    }
}
```

Implementare propriu zisă a clientului în coadă pentru coada cu numărul de clienți minim.



Clasa Server cu toate câmpurile și metodele aferente.



Clasa Task cu toate câmpurile aferente.

5. Rezultate

Pentru testarea acestei aplicații s-au introdus niște seturi de date.

Fișierul .log pentru unul din seturile de date:

Time: 0

Waiting clients: (0, 5, 2) (3, 7, 3) (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 1

Waiting clients: (0, 5, 2) (3, 7, 3) (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 2

Waiting clients: (0, 5, 2) (3, 7, 3) (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 3

Waiting clients: (0, 5, 2) (3, 7, 3) (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 4

Waiting clients: (0, 5, 2) (3, 7, 3) (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 5

Waiting clients: (3, 7, 3) (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb: (0, 5, 2)

QueueModel.Server@3f99d53c:

Time: 6

Waiting clients: (3, 7, 3) (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 7

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb: (3, 7, 3)

QueueModel.Server@3f99d53c:

Time: 8

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 9

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 10

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 11

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 12

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 13

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 14

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 15

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 16

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 17

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 18

Waiting clients: (2, 19, 3) (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 19

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb: (2, 19, 3)

QueueModel.Server@3f99d53c:

Time: 20

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 21

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 22

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 23

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 24

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 25

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 26

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 27

Waiting clients: (1, 28, 2)

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 28

Waiting clients:

QueueModel.Server@2a145deb: (1, 28, 2)

QueueModel.Server@3f99d53c:

Time: 29

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 30

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 31

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 32

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 33

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 34

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 35

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 36

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 37

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 38

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 39

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 40

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 41

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 42

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 43

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 44

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 45

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 46

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 47

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 48

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 49

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 50

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 51

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 52

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 53

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 54

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 55

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 56

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 57

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 58

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Time: 59

Waiting clients:

QueueModel.Server@2a145deb:

QueueModel.Server@3f99d53c:

Peak hour: 5

Average waiting time: 0.25

Average service time: 2.5

6. Concluzii

Din implementarea acestei aplicații am învățat lucrul cu fire de lucru în Java, pentru eficientizarea aplicațiilor.

7. Bibliografie

http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html -
http://www.tutorialspoint.com/java/util/timer_schedule_period.htm -
http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and threadpoolexecutor.html