DOCUMENTATIE

TEMA 2

NUME STUDENT: PEPELEA IOANA-ADINA

GRUPA: 30228

CUPRINS

1. Obiectivul temei 3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3
3. Proiectare 4
4. Implementare 6
5. Rezultate 9
6. Concluzii 10
7. Bibliografie 11

1.Obiectivul temei

Obiectivul principal al temei este de a realiza un sistem care poate gestiona un anumit numar de cozi, astfel incat sa minimizeze timpul de asteptare. Sistemul trebuie sa simuleze venirea unei serii de N clienti, care se aseaza la Q cozi, asteptand sa fie serviti si apoi parasind coada. Totodata, aplicatia trebuie sa calculeze si sa afiseze timpul mediu de asteptare, timpul mediu de servire si ora de varf.

Obiectivele secundare sunt urmatoarele:

-Analiza problemei si identificarea cerintelor

-Realizarea design-ului aplicatiei de simulare a cozilor

-Implementarea aplicatiei de simulare a cozilor

-Testarea aplicatiei pe baza unor input-uri prestabilite.

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

Analizand cerinta problemei, se pot deduce o serie de cerinte functionale si cerinte non-functionale care trebuie indeplinite.

Cerintele functionale sunt indeplinite de:

-Aplicatia de simulare ar trebui sa permita utilizatorilor sa configureze simularea

-Aplicatia de simulare ar trebui sa permita utilizatorilor sa inceapa simularea

-Aplicatia de simulare ar trebui sa afiseze cozile in timp real.

Cerintele non-functionale sunt:

-Aplicatia de simulare ar trebui sa fie intuitiva si usor de realizat.

Utilizarea aplicatiei in lumea reala, poate da nastere mai multor scenarii, care trebuie prevazute inca din faza de proiectare. Aceste cazuri de utilizare pot fi descrise cu ajutorul unor documente use-case, in care e descrisa evolutia sistemului si interactiunea utilizatorului cu acesta.

Use-case: Simularea unui sistem compus din Q cozi si N clienti

Primary actor: User

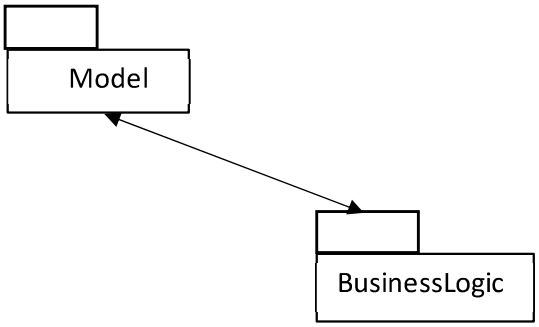
Main success scenario:

1) User-ul introduce datele necesare simularii: numarul de clienti(numberOfClients), numarul de cozi(numberOfServers), timpul maxim simularii(timeLimit), timpul minim si timpul maxim de sosire al clientilor(minArrivalTime si maxArrivalTime), timpul minim si timpul maxim de servire al clientilor(minProcessingTime si maxProcessingTime).
2) User-ul apasa butonul de Run
3) Se verifica daca datele introduse de utilizator sunt valide.
4) Se genereaza random un numar de N clienti, fiecare avand un id, un timp de sosire(arrivalTime) si un timp de servire(serviceTime), care vor fi afisati intr-o lista de asteptare, in ordinea crescatoare a timpului de sosire.
5) Se porneste simularea plecand de la timpul de simulare 0.
6) Se afiseaza N cozi, in care vor fi asezati clientii, pe masura ce timpul de simulare ajunge egal cu timpul de sosire al acestora.
7) Simularea se opreste in momentul in care timpul de simulare ajunge egal cu timeLimit.

Alternative sequence: User-ul introduce date despre simulare invalide. Aplicatia afiseaza un mesaj de eroare, prin care este semnalat motivul erorii, si prin care se cere introducerea altor date referitoare la simulare.

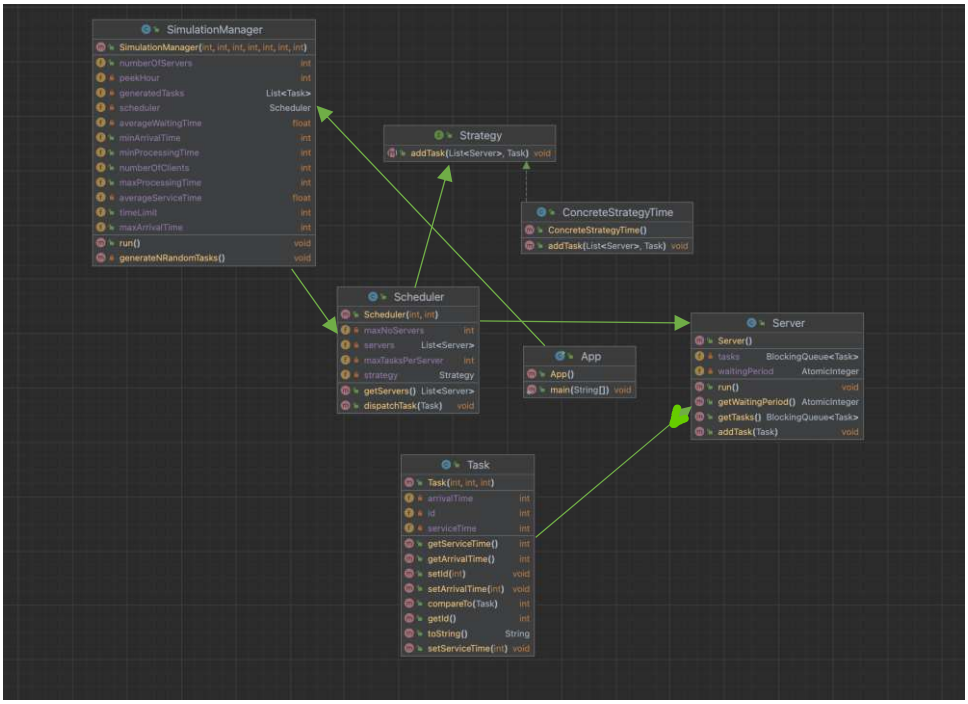
3. Proiectare

Luam in considerare cerintele functionale si cele non-functionale, proiectul va fi impartit in doua pachete: Model si BusinessLogic. Pachetul Model va contine doua clase: Task, care modeleaza un client cu atributele necesare(id, arrivalTime, serviceTime) si clasa Server, care implementeaza modelul unei cozi. Pachetul BusinessLogic contine 3 clase si o interfata: clasa ConcretStrategyTime care implementeaza interfata Strategy si implicit metoda addTask, clasa Scheduler in care ne cream cate un Thread pentru fiecare server, precum si serverele si clasa SimulationManager care implementeaza interfara Runnable, deci implicit si metoda Run(). Relatia dintre cele doua pachete este surprinsa in figura de mai jos:



Aplicatia este proiectata conform conceptelor de baza OOP. Abstractizarea presupune ca fiecare obiect are un rol bine stabilit, putand sa comunice cu celelalte obiecte, sa primeasca si sa furnizeze operatii, fara a da detalii despre implementarea facilitatiilor. Astfel, sistemul este compus din cinci clase, fiecare avand un rol foarte bine stabilit: Task, Server, ConcreteStrategyTime, Scheduler, SimulationManager. Incapsularea asigura ca obiectele nu pot schimba starea interna a altor obiecte decat prin metode de tip setter, toate atributele clasei avand modificatorul de acces private.

Relatiile dintre clasele definite in cadrul sistemului este surprinsa in diagrama UML de mai jos:



4. Implementare

a) Clasa Task este cea care reprezinta clientul ce trebuie sa ajunga in coada. Aceasta clasa implementeaza interfata Comparable, deci implicit implementeaza metoda compareTo(Object). Aceasta clasa are trei atribute: id, arrivalTime, serviceTime. Am implementat metoda toString() pe care am suprascris-o pentru a realiza afisarea task-ului. Metoda compareTo(Object) care primeste un parametru de tipul Object si pe care o folosim ca sa sortam crescator dupa arrivalTime lista de Task-uri generate random.

```
public class Task implements Comparable<Task>{
    8 usages
    private int arrivalTime;
    4 usages
    private int serviceTime;
    4 usages
    private int id;

    1 usage
    public Task(int arrivalTime, int serviceTime,int id) {
        this.arrivalTime = arrivalTime;
        this.serviceTime = serviceTime;
        this.id = id++;
    }
```

b) Clasa Server este cea care reprezinta coada la care trebuie sa ii adaugam clienti. Aceasta clasa implementeaza interfata Runnable, deci implicit si metoda run(). Aceasta clasa contine doua atribute: tasks de tipul BlockingQueue<Task> si waitingPeriod de tipul AtomicInteger. Am implementat metoda addTask(Task) care primeste un parametru de tipul Task, adauga in BlockingQueue-ul tasks acel task primit ca parametru. Metoda run() care cat timp tasks nu este goala, ia peek-ul din coada si daca acesta nu este null, adoarme Thread-ul curent si actualizeaza waitingPeriod-ul. Metoda getTasks() returneaza atributul tasks de tipul BlockingQueue al clasei Server. Metoda getWaitingfPeriod() care returneaza atributul de tipul AtomicInteger al clasei Server.

```java
public class Server implements Runnable{
    5 usages
    private BlockingQueue<Task> tasks;
    4 usages
    private AtomicInteger waitingPeriod;
    1 usage
    public Server(){
        this.tasks=new LinkedBlockingQueue<Task>();
        this.waitingPeriod=new AtomicInteger();
    }
    1 usage
    public void addTask(Task newTask) throws InterruptedException {
        try{
            tasks.put(newTask);
        }catch(InterruptedException ex){
            throw new InterruptedException();
        }
        waitingPeriod.incrementAndGet();
    }
```

C) Interfata Strategy care contine metoda addTask(List<Server>, Task) care este implementata de clasa ConcreteStrategyTime.

```java
public interface Strategy {
    1 usage  1 implementation
    public void addTask(List<Server> servers, Task t) throws InterruptedException;
}
```

d) Clasa ConcreteStrategyTime implementeaza interfata Strategy si implicit metoda addTask(List<Server>, Task). In aceasta metoda cautam coada cu cel mai mic waitingPeriod in care vom adauga Task-ul, adica clientul.

```java
public class ConcreteStrategyTime implements Strategy {
    1 usage
    public void addTask(List<Server> servers, Task t) throws InterruptedException {
        int min=3600, poz=0;
        for(int i=0;i<servers.size();i++){
            int value=servers.get(i).getWaitingPeriod().get();
            if(value<min){
                poz=i;
                min=value;
            }
        }
        servers.get(poz).addTask(t);
    }
}
```

e) Clasa Scheduler care contine 4 atribute: servers de tipul list<Server>, maxNoServers, maxTasksPerServer, strategy de tipul Strategy. In constructorul acestei clase ne cream cate un Thread pentru fiecare server, precum si serverele. Folosim metoda start() pentru Thread-urile

create. In aceasta clasa am implementat doua metode: metoda dispatchTask(Task) care primeste un parametru de tipul Task si care apeleaza metoda addTask(List<Server>, Task) pentru adaugarea in functie de strategia aleasa a clientilor in cozi. Metoda getServers() care nu primeste niciun parametru si care returneaza atributul servers al clasei Scheduler.

```
public class Scheduler {
    4 usages
    private List<Server> servers;
    1 usage
    private int maxNoServers;
    1 usage
    private int maxTasksPerServer;
    2 usages
    private Strategy strategy;
    1 usage
    public Scheduler(int maxNoServers, int maxTasksPerServer){
        this.maxNoServers=maxNoServers;
        this.maxTasksPerServer=maxTasksPerServer;
        this.servers=new ArrayList<Server>();
        for(int i=0;i<maxNoServers;i++){
            Server server=new Server();
            servers.add(server);
            Thread thread=new Thread(server);
            thread.start();
        }
        strategy=new ConcreteStrategyTime();
    }
```

f) Clasa SimulationManager implementeaza interfata Runnable, deci implicit si metoda run(). In aceasta clasa am implementat doua metode: metoda generateNRandomTasks() care genereaza o lista de tipul Task de dimensiune numberOfClients in care se adauga obiectele de tipul Task ale caror parametrii au fost generati random intre un range ce tine cont de datele introduse de utilizator. Tot aici am calculat averageWaitingTime si averageServiceTime. A doua metoda implementata este metoda run() pe care am suprascris-o si care pune clientii in cozi si ii sterge din lista generatedTasks. Tot aici verificam daca peek-ul fiecarei cozi are serviceTime-ul egal cu 0, in caz pozitiv, stergem Task-ul din coada respectiva si in caz contrar in scade. In aceasta metoda scriem in fisier rezultatele simularii.

```
public SimulationManager(int timeLimit, int maxProcessingTime, int minProcessingTime,
                         int maxArrivalTime, int minArrivalTime, int numberOfServers, int numberOfClients){
    this.timeLimit=timeLimit;
    this.maxProcessingTime=maxProcessingTime;
    this.minProcessingTime=minProcessingTime;
    this.maxArrivalTime=maxArrivalTime;
    this.minArrivalTime=minArrivalTime;
    this.numberOfClients=numberOfClients;
    this.numberOfServers=numberOfServers;
    this.peekHour=0;
    this.averageWaitingTime=0;
    this.averageServiceTime=0;
    this.scheduler=new Scheduler(numberOfServers, numberOfClients);
    this.generateNRandomTasks();
}
```

5. Rezultate

Ruland testele din tabelul furnizat in cerinta temei, a observat ca aplicatia functioneaza corect, furnizand totodata si timpul mediu de asteptare, timpul mediu de servire si ora de varf.

Test 1:

```
Timp simulare 52
Coada 1: []
Coada 2: []
Timp simulare 53
Coada 1: []
Coada 2: []
Timp simulare 54
Coada 1: []
Coada 2: []
Timp simulare 55
Coada 1: []
Coada 2: []
Timp simulare 56
Coada 1: []
Coada 2: []
Timp simulare 57
Coada 1: []
Coada 2: []
Timp simulare 58
Coada 1: []
Coada 2: []
Timp simulare 59
Coada 1: []
Coada 2: []
Peek hour is 12.
Average waiting time is 1.25.
Average service time is 2.5.
```

Test 2:

```
[(46,2,5), (42,4,3), (36,4,6), (23,4,2), (15,4,5), (2,4,5), (12,5,4), (29,7,6), (24,8,6),
Timp simulare 0
Coada 1: []
Coada 2: []
Coada 3: []
Coada 4: []
Coada 5: []
Timp simulare 1
Coada 1: []
Coada 2: []
Coada 3: []
Coada 4: []
Coada 5: []
Timp simulare 2
Coada 1: [(46,2,5)]
Coada 2: []
Coada 3: []
Coada 4: []
Coada 5: []
Timp simulare 3
Coada 1: [(46,2,5)]
Coada 2: []
Coada 3: []
Coada 4: []
Coada 5: []
```

Test 3:

```
Timp simulare 199
Coada 1: [(802,73,4), (410,75,5), (917,75,8), (107,76,5), (919,79,7), (124,81,5), (54,82,8), (688,84,4), (559,86,3), (2
Coada 2: [(475,70,6), (730,71,6), (771,73,5), (910,75,3), (101,76,5), (394,79,4), (605,79,4), (978,82,6), (574,83,3), (
Coada 3: [(548,66,6), (277,69,8), (86,69,4), (453,70,5), (672,73,8), (897,75,5), (31,76,7), (894,77,4), (543,79,3), (50
Coada 4: [(912,70,3), (398,70,6), (564,73,8), (696,74,5), (799,75,5), (424,77,3), (884,77,8), (765,80,7), (608,84,4), (
Coada 5: [(525,73,8), (683,74,4), (687,75,7), (659,77,5), (265,79,5), (757,80,5), (809,82,4), (563,84,5), (309,86,6), (
Coada 6: [(157,66,5), (750,69,5), (257,70,3), (825,72,4), (658,74,5), (638,75,7), (373,77,6), (643,77,3), (666,80,3), (
Coada 7: [(821,72,6), (657,74,5), (377,75,7), (627,75,4), (800,78,3), (636,80,5), (748,82,3), (472,84,5), (88,86,6), (3
Coada 8: [(131,70,5), (689,72,4), (613,74,8), (517,75,5), (279,77,6), (781,78,8), (467,80,8), (635,82,7), (325,84,5), (
Coada 9: [(596,69,4), (125,70,3), (603,74,4), (890,76,3), (210,77,8), (729,78,4), (400,80,5), (386,83,8), (207,84,4), (
Coada 10: [(333,72,7), (515,74,4), (753,76,6), (681,78,7), (214,79,6), (381,80,4), (204,84,7), (773,86,7), (994,87,5),
Coada 11: [(215,75,4), (732,76,8), (865,78,3), (667,78,4), (291,80,3), (601,82,3), (497,85,8), (256,85,8), (931,87,6),
Coada 12: [(867,70,3), (321,72,4), (294,73,3), (423,74,7), (728,76,5), (516,78,6), (379,81,8), (829,81,5), (785,84,4),
Coada 13: [(319,72,8), (367,74,8), (208,75,7), (726,76,3), (466,78,5), (775,81,8), (320,83,3), (862,85,3), (846,87,6),
Coada 14: [(941,72,7), (288,72,8), (343,74,3), (614,76,8), (448,78,6), (957,80,6), (591,82,8), (851,85,5), (489,87,3),
Coada 15: [(284,74,5), (252,75,3), (468,76,4), (861,78,7), (206,78,8), (197,81,4), (587,82,3), (746,85,3), (674,86,6),
Coada 16: [(121,72,4), (154,74,5), (165,75,8), (316,76,3), (201,78,7), (357,82,7), (164,83,7), (675,85,3), (553,87,5),
Coada 17: [(670,70,7), (49,72,8), (150,74,8), (44,75,3), (286,76,8), (74,78,7), (341,82,8), (40,83,8), (612,85,6), (530
Coada 18: [(911,74,4), (106,74,8), (274,76,8), (21,78,3), (56,79,6), (260,82,4), (767,83,8), (655,86,6), (925,88,4), (5
Coada 19: [(135,73,7), (878,74,8), (72,74,4), (245,76,5), (816,78,6), (37,79,3), (133,81,8), (149,82,7), (714,84,6), (4
Coada 20: [(541,70,8), (479,73,4), (23,73,3), (9,74,5), (186,76,7), (964,79,5), (928,80,4), (63,82,7), (690,84,3), (570
Peek hour is 99.
Average waiting time is 0.2703.
Average service time is 5.406.
```

6. Concluzii

Aceasta aplicatie a fost implementata cu ajutorul thread-urilor, ceea ce mi-a permis sa inteleg mai profund cum functioneaza si cum ar trebui manipulate.

Printre imbunatatirile pe care le-as aduce eu proiectului meu, ar fi adaugarea unei clase Service de unde sa se selecteze tipul de serviciu pe care clientii doresc sa-l primeasca. In functie de serviciul ales, interfata sa fie si ea schimbata, iar coada sa ii corespunda unui singur serviciu.

7. Bibliografie

https://dsrl.eu/courses/pt/materials/PT2023_A2_S1.pdf

https://dsrl.eu/courses/pt/materials/PT2023_A2_S2.pdf

https://www.w3schools.com/java/java_threads.asp