DOCUMENTATIE

TEMA 2

NUME STUDENT:Cândea Claudiu
GRUPA:30227

CUPRINS

1. Obiectivul temei

Obiectivul principal al temei este implementare unei aplicatii ce simuleza un sistem de cozi. Avem un numar de N clienti care intra in Q cozi, asteapta sa fie serviti si in final parasesc coada.

Obiective secundare:

a) Analiza problemei si identificare cerintelor
b) Proiectare simulatorului
c) Implementare simulatorului
d) Testare simulatorului

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

Cerintele functionale ale proiectului sunt:

-Simulatorul ar trebui sa ii permita utilizatorului sa initializeze simularea;

-Simulatorul ar trebui sa ii permita utilizatorului sa porneasca simularea;

-Simulatorul ar trebui sa afiseze in timp real evaluarea cozilor;

Cerintele non-functionale ale proiectului sunt:

-Aplicatia ar trebuie sa fie intuitiva si usor de folosit de catre utilizator;

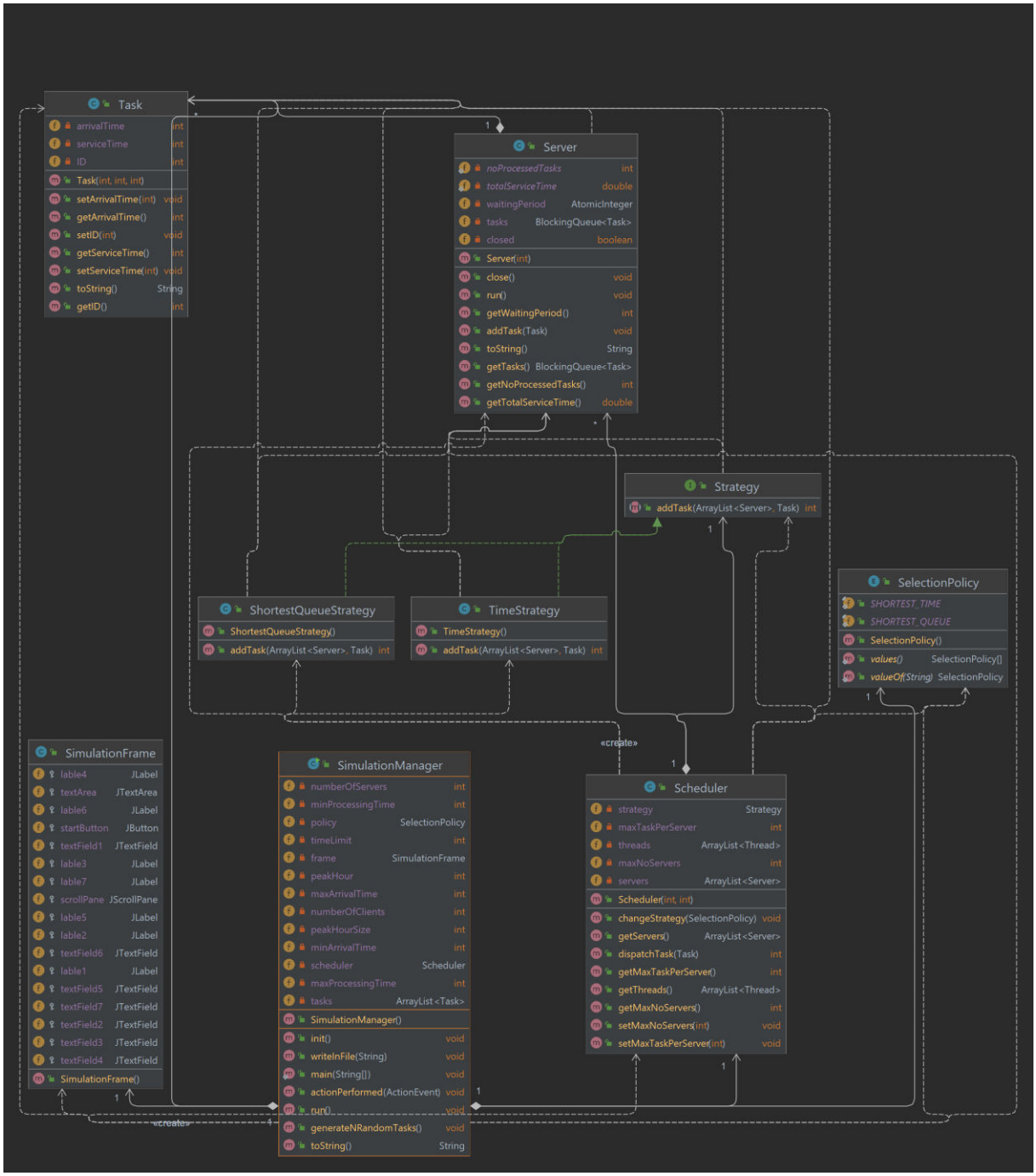
Use Case: simulare cu N clienti si Q cozi

Actor principal: Utilizatorul;

Scenariu de success:

1. Din interfata grafica utilizatorul introduce numarul de clienti N, max arrival time, min arrival time, max service time, min service time, timpul total de simulare si numarul de cozi Q, apoi va apasa butonul START;
2. Simulatorul va generea N clienti, arrival time si service time pentru fiecare client for fi generate aleator in functie de datele furnizate de utilizator;
3. In interfata grafica se va afisa evolutia simularii: la fiecare timp al simularii se for afisa clientii care inca nu sunt inclusi intr-o coada si cozile impreuna cu clientii care asteapta la aceastea;

4. Proiectare



Clasele Server si SimulationManager implementeaza interfata Runnable pentru a putea lucra cu threaduri, in cadrul implementarii fiecare coada(server) va rula pe un thread separate. De asemenea clasa SimulationMananger implementeaza si interfata ActionListener.

Am create si o interfata noua, Strategy, ce este implementata de Clasele TimeStrategy si ShortestQueueStrategy, acestea descriu modul in care este aleasa coada in care se adauga urmatorul task.

In clasa Server am folosit structura de date BlockingQueue pentru a implementa o coada de taskuri ce asteapta sa fie procesate, s-a ales aceasta structura pentru ca este

thread safety. De asemenea am mai folosit si ArrayList-uri, precum in clasa Scheduler care are un o lista de servere si una de threaduri pe care le gestioneaza. Si in clasa SimulationManager am folosit un ArrayList pentru a reprezenta lista cu toate taskurile generate la inceputul simularii.

5. Implementare

Clasa Task

Aceasta clasa descrie un Client al magazinului. Acesta are un ID, un timp de sosire la coada si un timp de servire(ID, arrivalTime, serviceTime) toate definite pe tipul int. Clasa are definite da metode doar getters and setters.

Clasa Server

Aceasta clasa implementeaza conceptul de coada al magazinului. Are ca variabile instanta un BlockingQueue de Taskuri(tasks), reprezentand lista clientilor ce asteapta la acea coada si un AtomicInteger, waitingPerioda, representant perioada de asteptata la coada. De asemenea, Clasa Server are si 3 variabile statice: variabila booleana closed, indicand daca se server-ul e deschis sau nu, variabila double totalServiceTime, in care se va calcula timpul de servire total pentru tot clientii indifferent de coada la care asteapta, si variabila int noProcessedTasks, ce indica numarul total de clienti ce au fost serviti la toate cozile.

Clasa are metode de get a variabilelor si metoda sincronizata addTask ce permite adaugare unui nou task in coada.

Metoda run ruleaza atat timp cat server-ul e deschis si extrage task-ul din capul cozi pe care il si procesaza: creste numarul taskurilor procesate, aduna serviceTime-ul acestuia la totalServiceTime si scade waitingPeriod. De asemenea, thread-ul va fi pus pe sleep un timp egal cu serviceTime-ul tasku-ului curent ce se proceseaza.

Clasele TimeStrategy si ShortestTImeStrategy

Ambele clase implementeaza interfata Strategy si implicit metoda addTask, ce primeste un ArrayList de servere si un task ce trebuie introdus in unul dintre acestea. Clasa TimeStrategy parcurge lista de servere si mai apoi adauga task-ul primit ca parametru in server-ul cu waitingPeriod-ul cel mai mic. Clasa ShortestQueueStrategy parcurge lista de servere sim ai apoi adauga task-ul primit ca si parametru in server-ul cu cele mai putine task-uri. Ambele salveaza waitingPeriod-ul pentru server-ul in care va fi adaugat task-ul si il returneaza.

Clasa Scheduler

Aceasta clasa are rolul de a organiza modul in care clientii sunt pusi in cozi. Ca variabile instanta are ca tip int maxNoServers si maxTaskPerServer, o instanta a interfetei Strategy in care se va pune un obiect din clasele ce o implementeaza, si 2 ArrayList, unul de tipul Server si unul de tipul Thread. Fiecare coada(server) va fi un thread seaparat create in costructorul acestei clase.

Metoda changeStrategy selecteaza in functie de argumentul policy ce tip de strategy ce va aplica(TimeStrategy sau ShortestQueueStrategy).

Metoda dispatchTask apeleaza metoda addTask din strategy, primind ca paramentru un Task si returneaza valoare de tip int primita in urma efectuarii metodei addTask.

Clasa mai are definite getters si setters.

Clasa SimulationFrame

Extinde clasa JFrame si reprezinta interfata grafica a aplicatiei. Are 7 JLabel-uri, 7 JTextField-uri, 1 JButton si un JTextArea atasat unui jScrollPane.

Din interfata se pot introduce datele initiale ale simulari si , apasand butonul, se poate porni simulare. In textArea se vor afis loguri in timp real cu evolutia simularii.

Clasa SimulationMananger

Aceasta clasa are rolul de a organiza intreaga simulare. Ca variabile instanta are: 1 scheduler, 1 SimulationFrame, 1 ArrayList de task-uri, 1 SelectionPolicy, variabile de tip int pentru max si min ArrivalTime, max si min ServiceTime, numarul de client si de servere, timpul total al simulari, ora de varfa a simularii si numarul total de taskuri din servere la ora de varf.

Aceasta clasa contine metoda main care porneste programul, metoda care instantiaza un obiect din clasa SimulationManager. Constructorul clasei initializeaza doar iterfata grafica a aplicatie (frame) si adauga obiectul curent (this) ca ActionListener pentru buttonul din interfata.

Metoda actionPerformed apeleaza metoda de init din SimulationManager, la apasarea butonului din interfata, si apoi creaza un thread, care reprezinta thread-ul principal al aplicatie, folsind obiectul din clasa SimulationManager cu cate a fost apelata metoda.

Metoda init intitializeaza varabilele obiectului SimulationManager, care nu au fost initializate in constructor, citind datele de intrare din interfata grafica. Apeleaza si metoda generateNRandomTasks din SimulationManager.

Metoda generateNRandomTask genereaza un numar de task-uri egal cu numarul de client introdus in interfata. Pentru fiecare task ArrivalTime si ServiceTime sunt generate random folosind valorile de maxim si minim ale acestor introduse in interfata.

Metoda run parcurge fiecare timp al simulari si adauga task-urile generate random in servere cand timpul simularii este egal cu ArrivalTime-ul pentru fiecare task. De asemenea, afiseaza la fiecare pas log-uri cu evolutia simularii in interfata grafica si intr-un fisier text. Cand s-a atins timpul total al simulari atunci calculeaza average Waiting Time si averga Service Time, pe care le v-a afisa in intefata grafica si in fisierul text alaturi de ora de varfa a simularii(peak hour).

6. Rezultate

Aplicatia a fost testata folosind-se urmatoare 3 seturi de date de intrare introduse din interfata grafica:

1)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]

2) 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]

3) 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]

Rezulatele acestor 3 simulari au fost introduse intr-un fisier text ce se v-a ragasi alaturi in folder-ul proiectului.

7. Concluzii

In urma acestei temea am invatat cum se lucreaza cu instantiezi un thread si cu se lucreaza cu acestea. Am inteles utiliatea majora ce care thread-urile o aduc in dezvolatea unei aplicatii software. De asemenea, am aprofundat cunostinte legate de cozi, creand o simulare pentru o aplicatie real a acestui concept.

Ca dezvolate ulterioara a acestui proiect s-ar putea imbunatatii interfata grafica si pentru fiecare coada/server waitingPeriod sa se decremezenteze cu 1 odata cu cresterea timpului de simulare.

8. 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-andthreadpoolexecutor.html

- https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf