

FUNDAMENTALS PROGRAMMING
TECHNIQUES
ASSIGNEMENT2
QUEUES SIMULATOR

Documentatie Tema2

Tincu Diana
An2 CTI Romana
Grupa 30221
AN 2020-2021

1. Obiectivul temei:

Se va proiecta si se va implementa simulator de cozi. Aplicatia va simula o serie de N clienti care ajung la cozi Q cozi , intra intr-una dintre cozi, si asteapta sa fie serviti, la final parasind coada, facand loc pentru urmatorul client din coada. Fiecare client este reprezentat de un id (un numar cuprins intre 1 si N), timpul de sosire - $t(\text{arrival})$ - (reprezinta timpul din momentul simularii la care clientul va intra in coada) si timpul de servire - $t(\text{service})$ - care reprezinta intervalul sau durata necesara pentru servirea clientului , sau timpul de asteptare al clientului din momentul in care este in capul cozii. Clientii vor fi generati aleator si mai apoi adaugati la cozi in functie de timpul lor de sosire.

Aplicatia va urmari parcursul clientilor in cozi , si va alege cea mai buna modalitate de a adauga clientii la cozi din punct de vedere al timpului de servire a celorlalti clienti deja introdusi in coada , sau al lungimii cozilor.

Rezultatele simularii se vor afisa intr - un fisier si in interfata grafica in timp real.

Tot in fisier sau in interfata grafica vom afisa si o medie a rezultatelor obtinute a timpului de asteptare , a timpului de servire si a timpului din momentul in care cozile sunt cele mai aglomerate.

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

Cozile sunt adesea utilizate si in lumea reala, avand aplicatii deosebit de importante. Principalul obiectiv al unei cozi este acela de a oferi unui client un loc de asteptare pentru a fi servit. Pentru a gestiona cat mai bine o coada ne intereseaza sa minimizam timpul de asteptare al fiecarui client, astfel daca o coada este goala clientul se va indrepta catre aceasta. Cu cat avem mai multe cozi cu atat timpul de servire al fiecarui client va fi mai bun, dezavantajul care apare sunt costurile suplimentare de memorie.

Pentru rezolvarea problemei am creat Q cozi, fiecărei cozi i- am atribuit un thread care sa asigure sincronizarea si executarea mai multor evenimente deodata. Am reusit sa implementez totul fara sa folosesc cuvantul cheie synchronized.

Am generat aleator N Task-uri (adica N clienti) , a caror intrare in cozi am planificat-o folosind o clasa implementata de mine numita Scheduler, care alegea sau planifica cea mai buna coada din punct de vedere a timpului de servire sau a lungimii cozilor. Pentru alegerea cozii pentru un client am folosit Strategy Pattern si am creat doua strategii concrete: una in functie de timp, si alta in functie de lungime.

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Structurile de date utilizate: List si ArrayList pentru a stoca liste de cozi (Servere) si List pentru lista de clienti (Task-uri) generati aleator. Am folosit de asemenea si BlockingQueue > pentru cozile de clienti.

Am creat trei pachete pentru a reprezenta mai bine modelul arhitectural Model View Controller: controller (in care am stocat partea pentru Controller adica SimulationController), modell (in care

am stocat partea pentru Model si tot ce tine de programul efectiv de simulare a cozilor) si nu in ultimul rand view(pentru a stoca tot ce tine de design-ul interfetei anume SimulationView). Clasele le-am adaugat in pachete astfel:

- In controller am adaugat: SimulationController
- In modell am adaugat: clasa ConcretStrategyQueue, clasa ConcretStrategyTime, clasa FilesUtil, clasa Scheduler, enumeratia SelectionPolicy, clasa Server, clasa SimulationManager, clasa Task si interfata Strategy
- In view am adaugat: SimulationView

4. Implementare

Clasa ConcreteStrategyQueue (Model)

In aceasta clasa am implementat codul pentru strategia alegerii cozii in functie de lungimea cozii, adica de cati clienti se afla intr- o coada. Aceasta clasa implementeaza interfata Strategy deci asadar implementeaza codul pentru metoda addTask:

```
public void addTask(List<Server> servers, Task t)
```

Clasa ConcreteStrategyTime (Model)

In aceasta clasa am implementat codul pentru strategia alegerii cozii in functie de timpul de asteptare al fiecarui client dintr- o coada, facand asadar suma pe fiecare coada a acestor timpuri si alegand- o pe cea mai mica, astfel incat clientul ce urmeaza sa fie adaugat sa astepte cat se poate de putin timp pentru a ajunge in capul cozii si pentru a fi servit. Aceasta clasa implementeaza interfata Strategy deci asadar implementeaza codul pentru metoda addTask:

```
public void addTask(List<Server> servers, Task t)
```

Clasa FilesUtil (Model)

In aceasta clasa am implementat o metoda care scrie intr- un fisier cu numele „ threadOutput.txt ” datele simularii in timp real. Pentru implementarea acestei metode am folosit metode si obiecte de tip BufferedWriter din bibliotecile java.io.BufferedWriter si java.io.FileWriter. Pentru a scrie in fisier am creat un obiect de tip BufferWrider, am creat un string care sa fie de forma celui pe care dorim sa il afisam , adica sa contina timpul curent al simularii, clientii care asteapta sa intre in coada (a caror t(arrival) este mai mic decat timpul curent de simulare currentTime), si pe urma pe randuri noi fiecare coada cu clientii sai la momentul curent al simularii.

Acestui string i- am facut append in fisier folosind apelul metodei implementate deja in java astfel:

```
writer.append(s);
```

Astfel fisierului in fisier nu se vor suprascrie datele de mai sus ci se vor adauga doar la finalul fisierului, pastrand toate datele simularii.

Clasa Scheduler (Model)

In aceasta clasa creez metodele care apeleaza alegerea pentru fiecare client a strategiei si a cozii in care urmeaza sa fie introdus.

Aceasta clasa are ca si instante: o lista de servere (cozi), numarul maxim de servere (cozi), numarul maxim de task- uri (clienti) dintr- un server (dintr- o coada) si strategia aleasa in functie de care se face alegerea cozii in care urmeaza sa intre clientul a carui timp de sosire este egal cu timpul de simulare curent.

Enumeratia SelectionPolicy (Model)

Aceasta contine strategiile implementate mai sus.

```
public enum SelectionPolicy {  
    SHORTEST_QUEUE, SHORTEST_TIME  
}
```

Clasa Server (Model)

Clasa Server implementeaza interfata Runnable intrucat pentru fiecare server (coada) am creat un thread.

Aceasta are ca si instante :

- o coada de task- uri (clienti)

```
private BlockingQueue<Task> tasks;  
- o perioada de asteptare
```

```
private AtomicInteger waitingPeriod;  
- timpul petrecut in coada
```

```
private int timeInQueue=0;  
- timpul de servire din coada
```

```
private int timeOfServiceInQueue=0;
```

In aceasta clasa avem implementata metoda addTask cu ajutorul careia adaugam task-urile in coada si calculam waitingPeriod- ul pentru coada respectiva.

Clasa SimulationManager (Model)

In aceasta clasa implementam metoda pentru generarea aleatoare a clientilor (task- urilor) si sortarea clientilor in functie de timpul lor de sosire in coada, tot in aceasta clasa avem un constructor SimulationManager() care ia din interfata creata datele necesare simularii si le proceseaza creand astfel lista de clienti apeland metoda prezentata anterior generateRandomTasks, si cozile apeland constructorul Scheduler(nr_cozi, nr_max_clienti_per_queue) si alege strategia care o vom folosi pentru adaugarea in cozi a clientilor.

Tot in aceasta clasa am creat si urmatoarele metode:

```
public void calcAverageWaitingTime() throws IOException
```

(aceasta metoda calculeaza timpul mediu de asteptare a unui client in coada)

```
public void calcAverageServiceTime() throws IOException
```

(aceasta metoda calculeaza timpul mediu de servire a unui client din toate cozile)

```
public void peakHour()
```

aceasta metoda calculeaza timpul la care este cea mai mare aglomeratie in cozi

```
public String updateFrame(int currentTime, List<Task> waitingTasks, List<Server> queue) throws IOException
```

(aceasta metoda face update la interfata creata pentru a afisa actualizarile din cozi la fiecare timp curent al simularii)

Aceasta clasa implementeaza si ea interfata Runnable deci am implementat in interiorul ei si metoda run(). In aceasta metoda am implementat codul pentru simularea cozilor in sine adica pentru introducerea clientilor in cozi atunci cand timpul de sosire este egal cu timpul curent de simulare si de scoatere din cozi atunci cand a trecut timpul de procesare al clientului din capul cozii. Tot aici am scris si codul pentru actualizarea listei clientilor care asteapta sa intre in cozi. Astfel, daca un client a intrat intr- o coada va fi scos din lista celor care asteapta sa intre in coada. Toate aceste modificari le- am afisat atat in fisierul creat cat si in interfata creata.

Clasa Task (Model)

Aceasta clasa reprezinta un client, ea are ca si variabile instanta un id de tipul int, un timp de sosire de tipul int, un timp de procesare sau de servire de tipul int si timpul de finalizare tot de tip int.

```
private int id;  
private int arrivalTime;  
private int finishTime;  
private int processingPeriod;
```

In aceasta clasa am creat doar cativa setteri si getteri pentru variabilele instanta si am creat o metoda toString care sa afiseze clientii in format String precum ex: (1 , 2 , 4) acesta este un client care are id-ul egal cu 1, timpul de sosire egal cu 2 si timpul de procesare sau de servire egal cu 4. Aceasta metoda o folosesc cand scriu in fisier sau in interfata grafica . Acesti clienti vor fi generati random si vor fi adaugati mai apoi in cozi alese dupa strategia selectata.

Clasa SimulationController (Controller)

In aceasta clasa am implementat o subclasa SimulateListener care implementeaza interfata ActionListener si care creaza ActionListener pentru butonul de simulare din interfata.

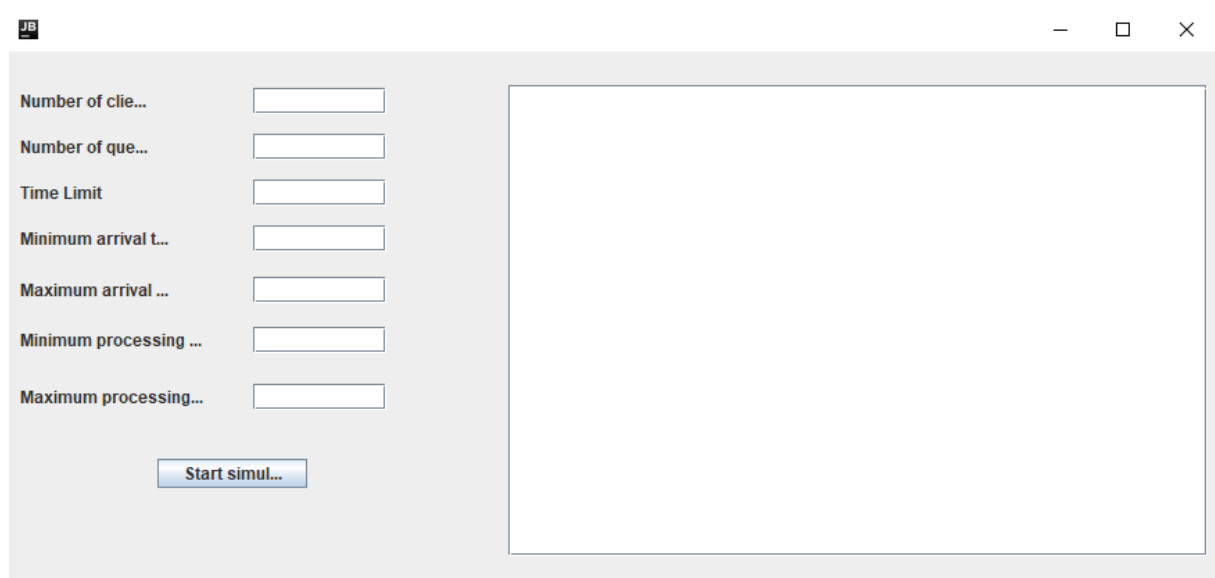
Aici am creat o variabila startSim care va fi egala cu 0 daca butonul de simulare din interfata creata nu a fost apasat sau va fi egala cu 1 daca butonul de simulare din interfata creata a fost apasat, adica daca am inceput sau nu simularea. Acest buton dupa cum am spus are rolul de a verifica daca a inceput sau nu simulare ca sa stim cand putem lua din casutele de text din interfata datele pe care le- am introdus . Daca nu am avea acest buton ar aparea niste erori pe parcursul rularii programului sau niste exceptii datorita faptului ca nu avem de unde extrage datele pentru a crea cozile si pentru a genera clientii si deci acestea din urma nu se vor crea si vor ramane nule sau neinitializate putand fi posibile totusi afisari incomplete, fara sens, sau cu lipsuri. Asadar , este foarte important sa avem in vedere

daca simularea a fost pornita sau nu , pentru a putea extrage corect datele din casutele de text din interfata grafica.

Clasa SimulationView (Controller)

In aceasta clasa, am creat design- ul interfetei , am adaugat la aceasta butonul de simulare, panoul pe care va urma sa afisam rezultatele obtinute in urma apasarii butonului de simulare, si sapte casute de text in care sa introducem datele necesare simularii si anume: numarul de clienti care trebuie generati, numarul de cozi disponibile , timpul limita sau timpul maxim de simulare (pana cand se executa simularea), timpul minim de sosire al clientilor care urmeaza sa fie generati , timpul maxim de sosire al clientilor care urmeaza sa fie generati, timpul minim de servire sau de procesare al clientilor care urmeaza sa fie generati , timpul maxim de servire sau de procesare al clientilor care urmeaza sa fie generati.

Pentru panoul pe care urmeaza sa afisam rezultatele simularii am adaugat si un scrollBar pentru a putea face vizibila intreaga simulare. Asadar, interfata creata de mine va arata asa:



5. Rezultatele testarii

Voi atasa un document separat cu rezultatele obtinute in urma simularii pe datele de intrare:

N=4,Q=2
tLimit=60sec
tMinArrivalTime=2
tMaxArrivalTime=30
tMinProcessingTime=2
tMaxProcessingTime=4

N=50,Q=5
tLimit=60sec
tMinArrivalTime=2
tMaxArrivalTime=40
tMinProcessingTime=1
tMaxProcessingTime=7

N=1000,Q=20
tLimit=200sec
tMinArrivalTime=10
tMaxArrivalTime=100
tMinProcessingTime=3
tMaxProcessingTime=9

6. Concluzii

În concluzie, am reușit să implementez un simulator de cozi pentru clienți generați. Iar pentru aplicația realizată am creat și o interfață foarte prietenoasă pentru a ajuta utilizatorul să introducă datele și să pornească simularea apăsând pe un buton ca mai apoi să poată să vadă rezultatele obținute în urma scurgerii timpului de simulare sau. Interfața este asadar foarte ușor de înțeles și de folosit. După cum știm deja cozile au o aplicație extrem de largă și în lumea reală, fiind ușor de înțeles și de aplicat. Asadar, am reușit să implementez simularea lor folosind Multithreading

7. Bibliografie

<https://www.w3schools.com/java/>

<https://www.oracle.com/ro/java/>