DOCUMENTAȚIE

TEMA *2*

NUME STUDENT: Coșarcă Ioan-Cristian

GRUPA: …30227……

CUPRINS

1. Obiectivul temei

Se va prezenta obiectivul principal al temei printr-o frază și un tabel sau o listă cu obiectivele secundare. Obiectivele secundare reprezintă pașii care trebuie urmați pentru îndeplinirea obiectivului principal. Fiecare obiectiv secundar va fi descris și se va indica în care capitol al documentației va fi detaliat.

Obiectivul acestui proiect / acestei teme este realizarea în Java a unei aplicații cu interfață utilizator dedicată pentru un sistem de gestiune a unor cozi care să implementeze mecanisme eficiente de alocare a cozilor, astfel încât să se evite timpi de așteptare mari și folosirea ineficientă a resurselor.

Aplicația va permite introducerea parametrilor de simulare, validarea lor și afișarea rezultatului pas cu pas de gestiune a cozilor, adăugarea și înlăturarea de clienți. De asemenea, la finalul simulării va afișa în cadrul interfeței grafice timpul mediu de așteptare, timpul mediu de servire și care a fost ora de vârf a cozilor.

Sub-obiective:

- Analiza problemei, modelare, scenarii, cazuri de utilizare (Capitolul 2)
- Proiectare (Capitolul 3)
- Implementare (Capitolul 4)
- Rezultate (Capitolul 5)

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

Se va prezenta cadrul de cerințe funcționale formalizat și cazurile de utilizare ca și diagrame și descrieri de use-case. Descrierile use-case-urilor se vor face sub forma unui flow-chart ori sub forma unei liste conținând pașii execuției fiecărui use-case.

Un program care să gestioneze modul de utilizare a unui număr dat de cozi poate reduce și rezolva problema de supraaglomerare a cozilor, timpi de așteptare lungi și folosirea ineficientă a resurselor.

Cerințe funcționale:

- Sistemul de gestiune al cozilor ar trebui să îi permită utilizatorului să seteze parametrii de simulare (număr de clienți, număr de cozi, timp total al simulării, parametrii timp de sosire și de servire)
- Sistemul de gestiune a cozilor ar trebui să îi permită utilizatorului să poată valida parametrii de simulare pentru a o putea apoi porni
- Sistemul de gestiune al cozilor ar trebui să îi permită utilizatorului să pornească simularea
- Sistemul de gestiune al cozilor ar trebui să îi permită utilizatorului să poată urmări evoluția în timp real a simulării
- Sistemul de gestiune al cozilor ar trebui să îi afișeze utilizatorului, după încheierea simulării, parametrii ce reprezintă timpul de așteptare mediu, timpul de servire mediu și ora de vârf care s-a observat în cadrul simulării

Cerințe non-funcționale:

- Sistemul de gestiune al cozilor ar trebui să fie intuitiv și ușor de folosit de către utilizator
- Sistemul de gestiune al cozilor ar trebui să îi permită utilizatorului să schimbe / modifice parametrii de simulare
- Sistemul de gestiune al cozilor ar trebui să poată reinițializa interfața pentru a afișa rezultatele între rulări și a nu le suprapune

Use Case: Inițializare Simulare

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul inserează în interfața grafică valorile pentru: numărul de clienți, numărul de cozi, intervalul de simulare (timpul maxim al simulării), minimul și maximul timpului de sosire și minimul și maximul timpului de servire.

2. Utilizatorul apasă butonul “VALIDATE” pentru a valida datele.

3. Aplicația de gestiune a cozilor validează datele și afișează un mesaj informându-l pe utilizator că poate da start simulării.

Secvență alternativă: Valori invalide pentru a inițializa simularea

- Utilizatorul inserează valori invalide pentru parametrii de pornire a aplicației.
- Aplicația afișează un mesaj de eroare și îi cere utilizatorului să insereze valori valide pentru parametrii.
- Scenariul se întoarce la pasul 1.

Use Case: Start și Urmărire Simulare

Actor Primar: Utilizatorul

Scenariu:

1. Odată ce valorile introduse în interfață au fost validate cu butonul “VALIDATE”, utilizatorul apasă pe butonul “START”.

2. Aplicația va genera atâția clienți caracterizați prin ID, timp de sorire și timp de servire, câți au fost specificați prin parametrul Număr Clienți introdus.

3. Aplicația va împărți clienții generați în numărul de cozi specificat la inițializare.

4. Aplicația de gestiune a cozilor va afișa în timp real evoluția cozilor și a servirii clienților.

5. După încheierea simulării, aplicația va afișa timpul mediu de așteptare al clienților în cozi, timpul mediu de servire și care a fost ora de vârf în cadrul simulării (la ce moment de timp s-au aflat cei mai mulți clienți în cozi).

3. Proiectare

*Se va prezenta proiectarea OOP a aplicației, diagramele UML de clase și de pachete, structurile de date folosite, interfețele definite și algoritmii folosiți (*dacă e cazul)

Proiectarea aplicației sistemului de gestiune a cozilor respectă arhitectura Model, View, Controller.

Clasa care reprezintă Controllerul este SimulationManager.

Vizualizarea și interfața sunt implementate în clasa SimulationFrame.

Componenta Model este alcătuită din clasele: Scheduler, Server și Task.

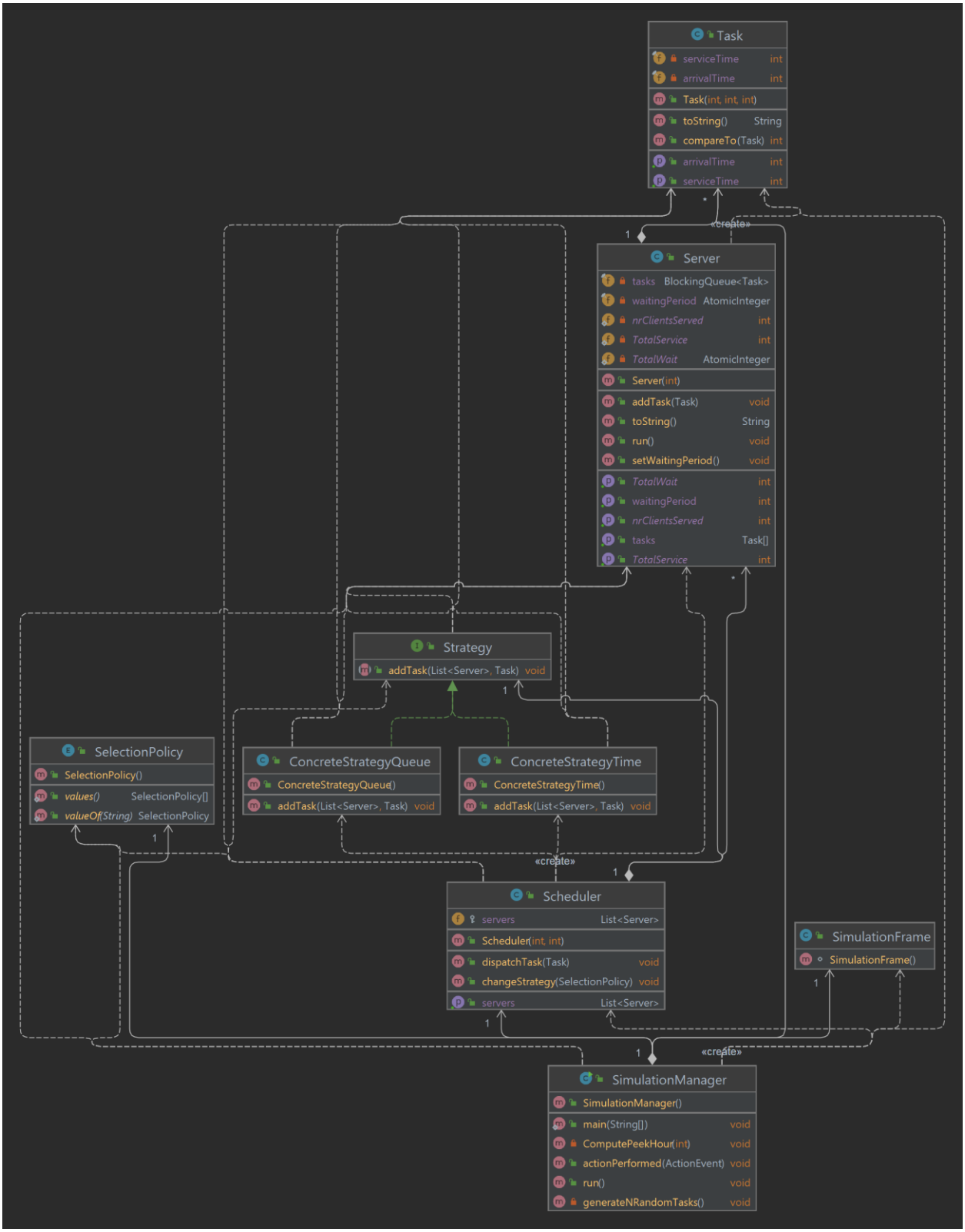
Interfața Strategy implementează metoda de adăugare a unui client într-o coadă. Această interfață este implementată în clasele ConcreteStrategyQueue și ConcreteStrategyTime care decid în ce fel se va face adăugarea în cozi ale clienților.

Enumerația SelectionPolicy specifică ce strategie să fie utilizată pentru selecția cozilor, având opțiunile SHORTEST_QUEUE și SHORTEST_TIME.

Diagrama UML de clase:

Ca și componente, aplicația este alcătuită din:

1. Clasele:
    - SimulationManager
    - ConcreteStrategyQueue
    - ConcreteStrategyTime
    - Scheduler
    - Server
    - Task
    - SimulationView
2. Interfața:
    - Strategy
3. Enumerația:
    - SelectionPolicy


Task
serviceTime int
arrivalTime int
Task(int, int, int)
toString() String
compareTo(Task) int
arrivalTime int
serviceTime int
Server
tasks BlockingQueue<Task>
waitingPeriod AtomicInteger
nrClientsServed int
TotalService int
TotalWait AtomicInteger
Server(int)
addTask(Task) void
toString() String
run() void
setWaitingPeriod() void
TotalWait int
waitingPeriod int
nrClientsServed int
tasks Task[]
TotalService int
Strategy
addTask(List<Server>, Task) void
SelectionPolicy
SelectionPolicy()
values() SelectionPolicy[]
valueOf(String) SelectionPolicy
ConcreteStrategyQueue
ConcreteStrategyQueue()
addTask(List<Server>, Task) void
ConcreteStrategyTime
ConcreteStrategyTime()
addTask(List<Server>, Task) void
Scheduler
servers List<Server>
Scheduler(int, int)
dispatchTask(Task) void
changeStrategy(SelectionPolicy) void
servers List<Server>
SimulationFrame
SimulationFrame()
SimulationManager
SimulationManager()
main(String[]) void
ComputePeekHour(int) void
actionPerformed(ActionEvent) void
run() void
generateNRandomTasks() void
«create»
«create»
«create»
1
*

Am folosit structurile de List și BlockingQueue pentru a reprezenta cozile, respectiv conținutul cozilor (ce clienți se află în cozi).

Clasele ConcreteStrategyQueue și ConcreteStrategyTime implementează interfața definită de utilizator Strategy, pentru a implementa metoda de adăugare a unui client în cozi, în funcție de strategia aleasă.

Clasele Server și SimulationManager implementează interfața Runnable și specifică, în metoda importată run, codul ce va fi executat de threadul clientului curent.

Clasa SimulationManager implementează interfața ActionListener pentru a specifica ce acțiuni să se execute în clipa în care, în cadrul interfeței, se apasă un anumit buton.

Clasa Task implementează interfața Comparable pentru a asigura ordonarea clienților după timpul lor de sosire.

4. Implementare

Se va descrie fiecare clasă cu câmpuri și metodele importante. Se va descrie implementarea interfeței utilizator.

Clasa ConcreteStrategyQueue:

Clasa adaugă un client în cozi în funcție de lungimea cozilor. Astfel, se calculează care este lungimea minimă curentă a cozilor și se adaugă clientul în coada cu lungimea respectivă.

Metoda addTask primește cozile / serverele și un client de adăugat / Task, caută unde să îl insereze și îl adaugă în coada respectivă.

Clasa ConcreteStrategyTime:

Clasa adaugă un client în cozi în funcție de timpul minim de servire curent al fiecărei cozi. Astfel, se calculează care este timpul de procesare minim și se adaugă clientul în coada cu timpul respectiv.

Metoda addTask primește cozile / serverele și un client de adăugat / Task, caută unde să îl insereze și îl adaugă în coada respectivă.

Clasa Scheduler:

Clasa decide ce procese să se execute și cât timp.

Clasa are ca și variabile: o listă de servere / cozi, numărul maxim de servere / cozi, numărul maxim de taskuri / clienți per server / coadă și strategia utilizată.

Metoda changeStrategy primește ca și parametru o politică de selecție și schimbă strategia care să fie aplicată în clipa când se vor construi cozile.

Metoda dispatchTask primește ca și parametru un client și apelează adăugarea acestuia în cozi, în funcție de strategia de inserare aleasă.

Clasa Server:

Clasa gestionează operațiile care să se execute asupra unui cozi.

Clasa are ca și variabile: o listă de taskuri / clienți, timpul de așteptare curent al cozii și variabilele statice de clasă: timpul de așteptare total al cozii, numărul de clienți serviți și timpul total de servire a clienților.

Metoda addTask primește un client, îl adaugă în cozi și actualizează variabilele instanță și de clasă.

În metoda run vom scoate un client din coadă, dacă aceasta nu este nulă.

Metoda getTasks construiește un vector de clienți ai cozii folosindu-se de variabila task și îl returnează.

Metoda setWaitingPeriod are rolul de a decrementa timpul curent de așteptare în coadă, în cazul în care coada nu e goală. Metoda va fi apelată pentru a decrementa treptat, la fiecare pas al simulării, timpul de așteptare dintr-o coadă.

Clasa Task:

Clasa construiește și reține informații despre un Client.

Clasa are ca și variabile: ID-ul clientului, timpul de sosire a clientului și timpul necesar al clientului pentru a fi servit.

Clasa SimulationManager:

Clasa comandă ce anume se va executa în program și ce anume se va afișa în interfața utilizator, prin receptarea introducerii de informații și apăsării de butoane de către utilizator.

Clasa are ca și variabile: timpul maxim cât va dura simularea, timpul maxim și minim de procesare, timpul maxim și minim de sosire a clienților în cozi, numărul de servere / cozi, numărul de clienți, politica de selecția a clienților în cozi, variabile de tipul Scheduler, Simulation Frame și variabilele statice de clasă: numărul maxim de clienți prezenți în cozi la un moment dat / dimensiunea maximă.

Metoda generateNRandomTasks generează un număr de clienți, specificat prin numberOfClients, clienți care vor fi caracterizați prin ID, timp de sosire (între timpul minim și timpul maxim), și timpul de servire necesar (între timpul minim și timpul maxim). După generarea tuturor clienților, pentru a descrie realist funcționarea cozilor și a simplifica selecția clienților, se sortează clienții crescător după timpul de sosire.

Metoda run pornește simularea, parcurge cozile, adaugă treptat clienții în cozi și îi scoate și calculează rezultatele simulării care vor trebui afișate.

Metoda ComputePeekHour primește timpul curent al simulării. Se calculează câți clienți se află în momentul prezent în cozi, iar dacă numărul e mai mare decât numărul de clienți deja găsit (sau 0), ora de vârf va fi egală cu timpul curent al simulare.

Metoda actionPerformed decide ce acțiuni să se producă în clipa când un buton al interfeței este apăsat. Dacă se apasă “Validate”, se citesc valorile parametrilor de simulare din

interfață și se caută validarea lor. Dacă nu sunt valizi, se cere reintroducerea lor, altfel se activează butonul “Start” ce va permite începerea simulării.

Clasa SimulationFrame:

Această clasă asigură implementarea interfeței grafice cu utilizatorul a aplicației.

Câmpurile care sunt declarate în clasă sunt elemente ale interfeței

Casetele de text permit introducerea informațiilor de simulare.

Casetele AvgWaitTime, AvgServiceTime, PHour vor afișa, la finalul simulării, datele despre timpul mediu de așteptare, timpul mediu de servire și ora de vârf.

În zona de text Log se vor înregistra și afișa în timp real pașii de simulare.

5. Rezultate

Se vor prezenta scenariile pentru testare cu Junit sau alt Framework de testare.

Pentru testarea aplicației de gestiune a cozilor vom rula 3 teste.

Se observă că pentru toate testele se începe prin a genera clienții. Pe urmă, se caută adăugarea lor într-o coadă.

Dacă o coadă este liberă, înaintea inserării va fi marcată cu “closed”.

Se observă la fiecare pas ce clienți se află la timpul curent în cozi, ce clienți se adaugă și care sunt clienți care sunt scoși.

Timpul mediu de așteptare care va fi calculat trebuie să fie cel puțin egal cu timpul mediu de servire.

Timpul mediu de servire va trebui să fie egal cu media aritmetică a tuturor timpilor de servire a clienților generați.

Ora de vârf va fi egală cu momentul de timp din cadrul simulării când s-au aflat cei mai mulți clienți în cadrul cozilor.

În urma rulării celor 3 teste, se observă că rezultatele obținute pentru cele 3 valori de ieșire respectă condițiile impuse.

6. Concluzii

Se vor prezenta concluziile, ce s-a învățat din temă, dezvoltări ulterioare.

În urma executării testelor, se constată că sistemul de gestiune al cozilor are comportamentul dorit. Se observă că aplicația calculează corect timpul mediu de așteptare în cozi, timpul mediu de servire și ora de vârf. De asemenea, sistemul gestionează bine adăugarea și scoaterea eficientă a unui client într-o coadă.

Se observă că adăugarea unui client în funcție de timpul de așteptare al cozii ține cont și de faptul că la momente de timp diferite clientul care se află în fruntea cozii are un timp de așteptare diferit până la terminarea procesării acestuia.

În urma acestui proiect m-am familiarizat cu lucrul cu Threaduri și cu gestiunea lor.

Ca dezvoltări ulterioare aș urmări îmbunătățirea stilistică a interfeței utilizator pentru a fi mai prietenoasă și mai aspectuoasă. De asemenea, aș urmări ca atunci când un client este procesat, pe măsura incrementării timpului total al simulării să se decrementeze timpul de servire al clientului, iar acest lucru să fie vizibil în interfață.

7. Bibliografie

https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html

https://www.baeldung.com/java-synchronized

https://www.baeldung.com/java-runnable-callable

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