DOCUMENTATIE TEMA 2

GENERARE COZI

Pop Alin

Grupa 30227

# Cuprins

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii
3. Proiectare
4. Implementare
5. Concluzii si posibile dezvoltari ulterioare

# Cerinte fundamentale

## Obiectivul principal

Proiectarea si implementarea unui program ce simuleaza si analizeaza evolutia cozile unui magazin, si genereaza staticii pentru aceaste date.

## Obiective secundare

* Dezvoltarea de use case-uri si scenarii
* Alegerea structurilor de date
* Impartirea pe clase
* Dezvoltarea algoritmilor
* Implementarea solutiei
* Testare

# 1.Analiza problemei

Utilizatorul trebuie sa introduca ca input datele necesare pentru rularea programului . Datele pe care utilizatorul trebuie sa le introduca sunt : intervalul dintre momentele in care poate sa ajunga un client nou , intervalul de timp necesar procesarii unui clinet , numarul de case la care poate fii distribuiti clientii si timpul cat va simula programul,

Scenariul in care programul va rula cu success :

* Utilizatorul introduce datele necesare pentru inceperea simularii in mod correct
* Utilizatorul apasa butonul de start pentru a incepe simularea
* Apare ecranul pentru simulare
* Pentru un timp egal cu timpul introdus de utilizator pentru simularea programului , programul va genera clienti noi si ii va distribui in cozi
* La final se vor afisa statisticile cerute

Scenarii in care programul nu va rula cu success :

* Utilizatorul nu va introduce date valide pentru inceperea programului , caz in care va aparea un mesaj de eroare de genul “Date invalide”

# Proiectare

Proiectul contine 7 clase acestea fiind :

* Task
* Server
* Scheduler
* ConcreteStrategyQueue
* Interface
* App
* Simulatonmanager

Mai contine si o interfata Strategy , aceasta interfata avand o singura metoda addTask1 , metoda care este implementata in clasa ConcreteStrategyQueue.

Proiectul este impartit in e pachete :

Pachetul Shop

* Aici regasim clasele Task si Server , acestea continand logica programului

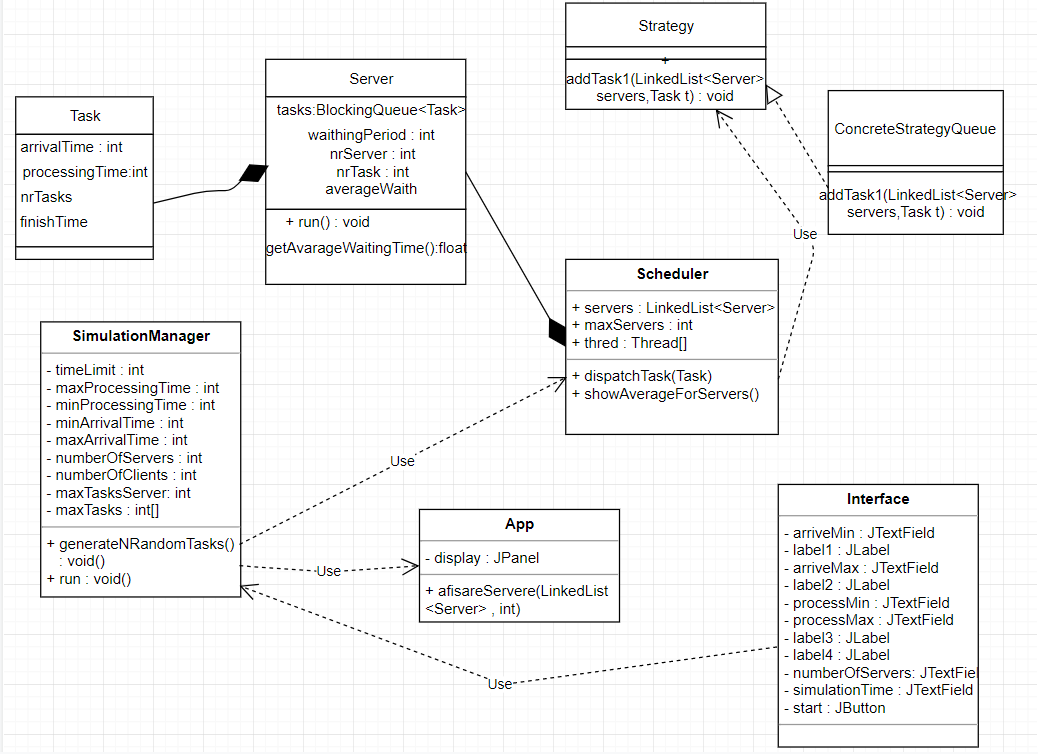
Pachetul Strategy

* Aici regasim interfata Strategy si clasele Scheduler,SimulationManager si ConcreteStrategyQueue unde este implementata logica dupa care sunt distribuiti clientii la cozi,

Pachetul userInterface

* Aici regasim clasele App si Interface care sunt destinate interfetei grafice pe care o va utiliza cel care ruleaza programul.

Diagrama UML:



# Implementare

# Task

In clasa Task avem 4 variabile de tip int , acestea fiind private , adica pot fi vazute doar in clasa Task . Aceste variabile sunt : arrivalTime , aceeasta reprezentand timpul la care clientul a fost distribuit uneia din cozi. processingTime , aceasta reprezentand timpul necesar pentru ca un client sa fie procesat dup ace a ajuns la coada . nrTask reprezinta numarul total de client . finishTime reprezinta momentul de timp in care clientul a plecat de la coada , acesta fiin repreentat de suma dintre arrivalTime si processingTime.

Metoda public int getNrTask() are rolul de a return numarul total de client . Metoda public int getArrivaltime() are rolul de a returna timpul la care un client a ajuns la coada , Metoda setArrivalTime( int t) are rolul de a seta timpul la care clientul la care se face referinta a ajuns la coada , cu valoarea t transmisa ca si parametru . Metoda getProcessingTime() are rolul de a returna timpul necesar procesarii unui client . Metoda setProcessingTime ( int t ) are rolul de seta timpul necesar procesarii unui client cu valoarea t transmisa ca si parametru . Metoda setFinihsTime (int i) are rolul de a seta timpul de finalizare al unui client cu valoarea I transmisa ca si parametru . Metoda getFinishTime are rolul de a returna timpul de finalizare al unui client .

Clasa Task implementeaza interfata Comparable<Task> si suprascrie metoda compareTo( Task t) . Aceasta metoda retine intr-o variabila arrivaTime-ul taskului transmis ca parametru , apeland metoda getArrivalTime , dupa care returneaza diferenta dintre arrivalTime-ul taskului pentru care a fost apelata si arrivalTime-ul taskului transmis ca si parametru . Rolul acestei emtode suprascrise este de a putea ordona taskurile dupa arrivalTime .

Constructorul clasei Task are rolul de a seta arrivaltime , processingTime si nrTask cu 3 variabile care sunt transmise ca si parametri.

# Server

In clasa Server regasim 5 variabile , 3 de tip int , una de tip float si una de tip BlockingQueue<Task>. Variabila tasks este un BlockingQueue < Task > si reprezinta practice o coada . waithingPeriod reprezinta timpul pe care un client il petrece pana sa ajunga in coada . nrServer reprezinta un indice care indica numarul cozii , nrTask este un indice care reprezinta numarul clientului , iar averageWaith reprezinta media timpului de asteptare pentru o coada .

Constructorul acestei clase are ca parametru o variabila de tip int , cu care initializeaza numarul de cozi , si creeaza in tasks un nou LinkedBlockingQueue < Task > .

Metoda addTask ( Task nou ) primeste ca parametru un Task , adauga la BlockingQueue-ul tasks clientul nou transmis ca si parametru , creste numarul de client pentru coada respective , dupa care adauga la waithingPeriod processingTime-ul taskului adaugat .

Metoda getNrQueue are rolul de a returna numarul de cozi . Metoda setNrQueue ( int n ) are rolul de a seta numarul de cozi cu valoarea parametrului n . Metoda setWaithingPeriod ( int I ) are rolul de a seta waithingPeriod – ul cu valoarea parametrului I . Metoda getWaithingPeriod are rolul de a returna valoarea lui waithingPeriod . Metoda getSize () are rolul de a returna marimea cozii . Metoda getTasks () care este de tip BlockingQueue<Task> are rolul de a returna valorile din coada .

Metoda public float getAvarageWaitingTime() are rolul de a returna timpul mediu de asteptare pentru coada pentru care este apelata .

Clasa Server implementeaza interfata Runnable si suprascrie metoda run . In aceasta metoda , cat timp coada nu este goala , se ia primul client din coada , I se seteaza timpul de finalizare cu timpul la care a ajuns + timpul de care e nevoie pentru a fi procesat . In averageWaith se retine timpul de asteptare pentru client . Se asteapta pentru un timp egal cu timpul de procesare al clientului , dupa care se afiseaza un mesaj cum ca , clientul a parasite coada , se actualizeaza waithingPeriod-ul si se sterge clientul din coada .

# Interfata Strategy

In inerfata strategy regasim o singura metoda addTask1 ( LinkedList < Server > , Task t ) . Aceasta metoda are rolul de a introduce clientii in cozi dupa o anumita strategie pe care o allege cel care implementeaza programul .

# ConcreteStrategyQueue

Clasa ConcreteStrategyQueue implementeaza interfata Strategy . Pentru a implementa interfata Strategy in aceasta clasa vom suprascrie metoda addTask1 . In aceasta metoda vom parcurge cu o variabila de la 0 pana la marimea cozii , si retine intr-o variabila min indicele cozii cu cei mai putini clienti . La final afiseaza un mesaj in care spun ca , clientul a fost introdus in coada , si adauga clientul transmis ca si parametru la coada cu cei mai putini client .

# Scheduler

In clasa Scheduler avem un LinkedList<Server> , adica o lista de cozi , o varibaila de tip int care reprezinta numarul maxim de cozi , si o variabila thred care este un sir de Threaduri .

In constructorul acestei clase initializam nuamrul amxim de cozi cu valoarea transmisa ca si parametru , creem o noua coada , si un nou sir de threaduri de marimea numarului amxim de cozi (creem cate un thread pentru fiecare coada ) . Parcurgem toate cozile , pentru fiecare coada apelam constructorul din server cu numarul cozii curente , adaugam serverul creeat in coada , creem un nou thead pentru coada creeata si il pornim.

Metoda dispatchTask ( Task c) apeleaza metoda addTask1 din interfata Strategy pentru coada si Clientul transmis ca parametru .

Metoda showAverageForServers parcurge toate cozile , si pentru fiecare coada afiseaza averageWaithingTime apeland metoda getAverageWaitingTime ( ) .

# SimulationManager

In aceasta clasa regasim toate datele necesare simulari acestea fiind : timLimit care repreinta timpul limita in care se face simularea , minProcessingTime si maxProcessingTime care reprezinta intervalul necesar pentru a procesa un clinet , minArrivalTime si maxArrivalTime care reprezinta intervalul necesar pentru ca un client sa ajunga la coada , numberOfServers care reprezinta numarul de cozi la care se pot aseza clientii , maxTaskServer care reprezinta numarul maxim de clineti care se pot aseza la o coada , si maxTasks care va fi folosit pentru a retine cati client se afla la toate cozile in intervalul timpului de procesare .

In constructorul acestei clase se vor asigna valori pentru fiecare din valorile necesare pentru a incepe procesul de simulate si se va instantia o variabila de tip Scheduler , va fi deschisa interfata pentru simulare si se va apela metoda generateNRandomtasks ( ) .

In metoda generateNRandomasks se va retine nuamrul clientilr , pentru processingTime si arrivalTime se va generale o valoare aleatoare intre minProcessingTime si maxProcessingTime respective minArrivalTime si maxArrivalTime . Se vor sorta clientii din coada dupa arrivalTime . Cat timp arrivalTime este mai mic decat timpul limita pentru generarea clientilor , se va creea un client nou , se va adauga clientul in coada , dupa care se vor actualize processingTime si arrivalTime .

Clasa SimulationManager implementeaza interfata Runnable , astfel ea trebuie sa suprascrie metoda run . De cate ori se va crea un thred nous i va fi pornit , se va apela metoda run care verifica daca timpul current este mai mic decat timpul simularii , daca este mai mic , daca avem un client care are arrivalTime-ul egal cu timpul current apeleaza medota dispatchTask pentru a pune clientul la o coada de asteptare . Dupa aceasta se apeleaza metoda afisareServere din clasa App pentru toate serverele si numarul acestora . Se incrementeaza curentTime , si se asteapta o secunda . La finalul metodei se fac demersurile pentru a afisa averageWaithingTime pentru fiecare server si peak Time – ul .

# App

Clasa App are o variabuila display de tip JPanel . in Constructorul clasei se initializeaza JPanelul si se fac setari corespunzatoare pentru acesta .

In aceasta clasa avem metoda afisareServere ( LinkedList < Server > , int nrServers) . Aceasta metoda sterge tot din panel si il revalideaza , dupa care creeaza un JScroll si un JList . Parcurge toate cozile , parcurge toti clientii din fiecare coada , si pentru fiecare client retine intr-o lista un mesaj de tipul “Clientul cu numarul : client.getNrTask“ , dupa care introduce acest mesaj in JScrolluri . La final pune in panelul de la inceput JScrollurile create cu mesajele aferente .

# Interface

In aceasta clasa se fac toate demersurile pentru a incepe procesul de simulare . Se creeaza text labeluri pentru a-I spune utilizatorului ce trebuie sa introduca pentru fiecare camp din TextFielduri . Astfel avem 6 labeluri si 6 textfielduri , pentru minArrivalTime , maxArrivalTime , minprocessingTime , maxProcessingTime , numberOfservers si simulationTime . Mai avem si un buton pentru a incepe simularea si panelul in care se afiseaza tot .

In constructorul aceste clase vom seta locul in care vrem sa apara fiecare element din interfata , adica textfieldurile si albelurile , precum si butonul de start . Tot in constructor vom face si un actionlistener pentru butonul de start , cu o clasa anonima , in care vom suprascrie emtoda actionPerformed , aici creeind o instant de tip Simulationmanager cu valorile pe care le vom introduce din interfata grafica , dupa care vom creea un thread nous i il vom porni . In cazul in care vom da peste orice tip de eorare vom afisa o caseta cu un mesaj de eroare . La final vom introduce toate componentele in labelul nostrum si il vom seta ca fiin labelul vizibil din frame .

La final am facut metoda main in care doar creem o instanta de tip Interface .

# Concluzii si metode de dezvoltare

Tema este una foarte interesanta si ajuta mult pentru a intelege programarea cu threaduri . Evolutia cozilor trebuie sa fie una concurenta si nu secventiala , astef ca este necesara creearea unui nou thread pentru fiecare coada . Programul trebuie sa ruleze pana in momentul in care timpul current atinge timpul limita introdus de catre utilizator , lucru care este posibil tot datorita threadurilor .

Proiectul poate fii inbunatatit prin afisarea mai multor statistici , sau spre exemplu deschiderea unor noi cozi in momentele de timp in care la fiecare coada numarul de client este mai mare decat un anumir numar . O alta dezvoltare posibila a proiectului este implementarea de strategii multiple pentru asezare a clientilor la coada , spre exemplu asezarea dupa timpul in care acestia au intrat in magazin.