**TEHNICI DE PROGRAMARE FUNDAMENTALE**



**ASSIGNMENT II:**

**QUEUE SIMULATOR**

**Documentatie**

STUDENT: MARCU MIHAI-ALEXANDRU

GRUPA: 30227

**CUPRINS:**

1. Obiectivul temei;
2. Analiza problemei, modelare, scenarii si cazuri de utilizare;
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator);
4. Implementare:

a. Clasa Task;

b. Clasa Server;

c. Clasa Scheduler;

d. Interfata Strategy;

e. Clasa ConcreteStrategyTime;

f. Clasa SimulatorManager.

1. Testare;
2. Concluzii;
3. Bibliografie.
4. Obiectivul temei:

Obiectivul principal al temei este realizarea unei aplicatii de simulare care are ca scop analizarea sistemelor bazate pe o coada pentru determinarea si minimizarea timpului de asteptare a clientilor.

Obiectivele secundare ale acestei teme sunt:

* Familiarizarea cu utilizarea thread-urilor si metodelor corespunzatoare acestora;
* Implementarea unui generator de clienti random;
* Realizarea unei Java Archive;
* Citirea datelor dintr-un fisier si afisarea rezultatelor obtinute in alt fisier, ambele fiind date ca parametrii in main.

1. Analiza problemei, modelare, scenarii si cazuri de utilizare:

In prima instanta am abordat problema modului in care functioneaza coada,in principiu modul in care clientii se pun la coada, problema pe care am rezolvat-o prin implementarea unui organizator care repartizeaza clientii la coada cu cel mai mic timp de asteptare si care stocheaza informatii despre fiecare aceasta. Datele de intrare pentru simularea proiectului vor fi citite dintr-un fisier, acestea fiind de forma:

* + Numarul de clienti;
  + Numarul de cozi;
  + Intervalul de simulare;
  + Valoarile minime si maxime pe care le poate lua timpul de sosire la coada a unui client;
  + Valoarile minime si maxime pe care le poate lua timpul de procesare a unui client.

Rezultatele obtinute vor fi afisate intr-un fisier de iesire.

Numele fisierului de intrare cat si al celui de iesire sunt date ca parametrii a metodei main. Cand programul va rula, acesta va crea un fisier cu numele primit ca al doilea parametru.

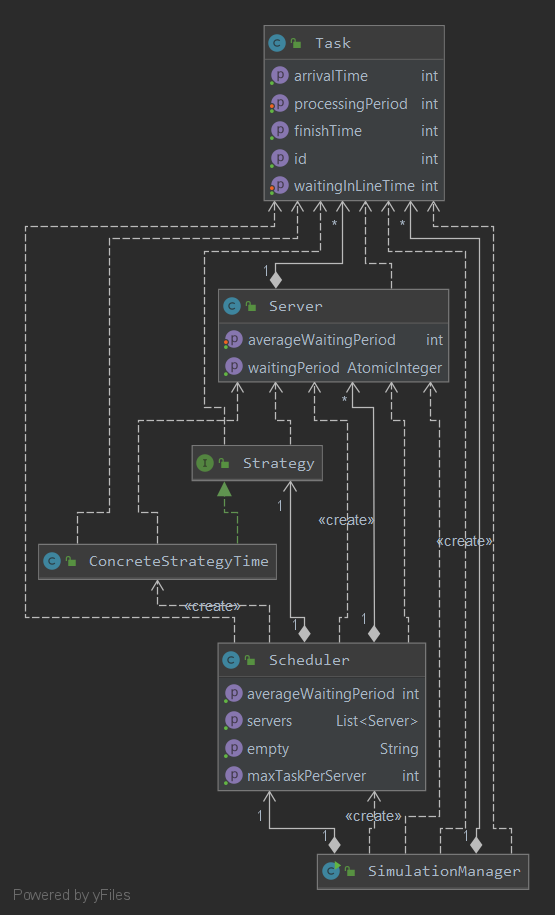
Pentru fisier este realizata o arhiva Jar care permite rularea programului cu cei doi parametrii din linia de comanda.

1. Proiectare(decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, pachete, algoritmi, interfata utilizator):

Pentru inceput am creat clasa Task care este reprezentativa unui client, care este caracterizat de catre un id si de catre trei perioade: prima este timpul din simulare in care clientul se pune la coada si al doilea este perioada de procesare care este reprezentata de catre timpul necesar servirii clientului atunci cand acesta este pozitionat la inceputul cozii, iar ultima perioada este timpul de finalizare care este calculata prin insumarea celor doua perioada mentionate anterior si perioada de asteptare a cozii la care clientul este situat din momentul respectiv. Dupa aceea am creat clasa Server care corespunde unei cozi si care este caracterizata de clientii care se afla in acea coada si de timpul de asteptare al cozii care este format din timpii de procesare ai fiecarui client aflat in acel moment in coada. Aceasta clasa a fost urmata de clasa numita Scheduler, care are rolul de a stoca informatii despre fiecare coada si de a repartiza fiecare client la coada cu cel mai mic timp de asteptare. Acesta este caracterizat de numarul maxim de cozi disponibile si de numarul maxim de clienti pe care o coada ii poate suporta. Dupa aceea am realizat clasa ConcreteStrategyTime care reprezinta strategia dupa care organizatorul repartizeaza clientii in cozi. Ultima clasa implementata a fost SimulationManager care are rolul de a realiza partea de simulare a proiectului. Dupa aceea am realizat arhiva Java.

Ca si structuri de date am utilizat: arrayblockingqueue si arraylist.

Diagrama UML a proiectului este urmatoarea:



1. Implementare:
2. Clasa Task: este reprezentativa unui client si contine campurile: id, arrivalTime, processingPeriod, waitingInLineTime si finishTime.

Metodele acestei clase sunt:

* Un constructor care are rolul de a initializa un client nou cu campurile date ca parametrii;
* Cate un setter si un getter pentru majoritatea campurilor clasei;
* Un compareTo care realizeaza sortarea clientilor crescator dupa timpul la care acestia se pun la coada;
* O metoda toString care are rolul de a transforma obiectul de tipul Task intr-unul de tip String si care va ajuta la scrierea in fisier dupa formatul specificat in cerinta temei.

1. Clasa Server: este rerprezentativa unei cozi. Campurile acestei clase sunt: tasks, de tipul Blockingqueue, care este coada in sine, waitingPeriod care este de tipul AtomicInteger si care este perioada de asteptare pana cand un client nou poate ajunge la casa(in varful cozii) din coada respectiva, un int numit averageWaitingPeriod care isi regaseste scopul in calcularea timpului mediu final de asteptare din coada si un boolean th care reprezinta conditia dupa care thread-ul cozii functioneaza.

Metodele clasei Server sunt urmatoarele:

* Un constructor care primeste ca parametru un int ce reprezinta numarul maxim de task-uri din coada. In acest constructor se initializeaza blockingqueue-ul ca un arrayblockingqueue de capacitatea data ca parametru si seteaza waitingPeriod cu valoarea initiala 0.
* Getter-e pentru waitingPeriod si averageWaitingPeriod si setter-e pentru averageWaitingPeriod, in care adun pentru fiecare client din coada respectiva timpul pe care l-a petrecut in aceasta de cand a sosit si pentru variabila th;
* Metoda addTask care are rolurile de a adauga un client in coada respectiva dar si de a incrementa perioada de asteptare din aceasta cu perioada de procesare a clientului introdus, pe care o adaug timpului mediu de asteptare cu ajutorul metodei de setter prezentata mai sus. In aceasta metoda mai calculez si timpul de finalizare a clientului si il setez pentru acesta si apelez notify() pentru a trezi thread-ul atunci cand un client este introdus in coada;
* suspendTh are rolul de a seta th cu valoare false si de a apela wait() pe thread-ul cozii pentru a-l suspenda cat timp coada este goala, fiind utilizata in metoda run;
* Metoda run: are scopul de a realiza functionarea thread-ului care are conditia de functionare th=true. Daca coada este goala atunci se apeleaza functia suspendTh, iar in caz contrar clientului din varful cozii ii este decrementata perioada de procesare pana aceasta devine 0, in acelasi timp thread-ul este pus la somn pentru 1 secunda, si cozii ii este de asemenea decrementata perioada de asteptare cu valoare perioadei de procesare a clientului respectiv. Dupa aceea metoda elimina clientul din coada si verifica daca aceasta a devenit goala, iar daca aceasta conditie este indeplinita este apelata functia suspendTh.
* O metoda toString care are rolul de a transforma obiectul de tipul Server intr-unul de tip String si care va ajuta la scrierea in fisier dupa formatul specificat in cerinta temei. Aceasta metoda returneaza string-ul „closed” daca coada este goala, iar in caz contrar string-ul returnat contine enumerati clientii din coada sub forma formatului cerut.

1. Clasa Scheduler: reprezinta organizatorul cozilor si are scopul de a mentine informatii despre fiecare coada si de a repartiza clientii din lista de asteptare la coada cu cea mai mica perioada de asteptare. Campurile acestei clase sunt: o lista de obiecte Server, doua int-uri care reprezinta numarul de cozi si numarul maxim de clienti pe care o coada ii poate contine, un string „empty” care va fi utilizat in clasa SimulationManager, un int in care se salveaza perioada medie de asteptare din toate cozile si un obiect numit strategy care apartine interfetei cu acelasi nume si are scopul de a stabili strategia dupa care sunt pusi clientii din lista de asteptare in cozi.

Metodele acestei clase sunt:

* Un constructor care primeste ca parametrii numarul de cozi si numarul de clienti pe care o coada ii poate contine. Acesta initializeaza lista de cozi ca un arraylist cu capacitatea egala cu primul parametru, dupa care initializeaza acel numar de obiecte de tip server pe care le adauga arraylist-ului si le creeaza fiecaruia un thread pe care il si porneste;
* Un getter pentru numarul de clienti maxim dintr-un obiect de tip Server, unul pentru string-ul empty si unul pentru lista de obiecte de tip Server;
* Metoda dispatchTask primeste ca parametru un obiect de tipul Task pe care il transmite mai departe metodei addTask care este apelata de obiectul strategy si este declarata in interfata Strategy si este implementata in clasa ConcreteStrategyTime;
* getAverageWaitingPeriod are scopul de a seta si returna timpul total de asteptare din toate cozile;
* Ultima metoda implementata in aceasta clasa este toString care ajuta la afisarea in fisier dupa formatul cerut si seteaza stringul empty cu „empty” in cazul in care toate cozile sunt goale sau „not empty” in caz contrar.

1. Interfata Strategy: in ea este declarata functia addTask;
2. Clasa ConcreteStrategyTime: implementeaza metoda addTask din interfata Strategy. Aceasta metoda are doi parametrii: lista cu obiecte de tip Server si un obiect de tipul Task si parcurge lista data de doua ori. Pentru inceput este initializat un int care reprezinta numarul minim de clienti pe care una dintre cozile din lista ii are. In prima parcurgere este cautata si setata aceasta valoare, iar in cea de a doua se apeleaza metoda addTask din clasa Server careia ii este dat ca parametru obiectul de tip Task, care a fost primit ca parametru in metoda care este prezentata acum, daca coada este goala sau daca are acelasi timp de asteptare egal cu valoare setata in parcurgerea precedenta si se realizeaza break pentru a iesi din parcurgere.
3. Clasa SimulationManager: are ca si campuri sapte int-uri care sunt citite din fisier si care reprezinta: numarul de clienti, numarul de cozi, numatul maxim si cel minim atat de ajungere cat si de procesare pe care un client poate sa le aiba si perioada de simulare a proiectului. De asemenea clasa mai contine si un float care reprezinta timpul mediu de asteptare final, un obiect de tipul Scheduler, o lista de obiecte de tipul Task si doua string-uri care sunt folosite pentru citirea si afisarea din fisier, in acestea sunt salvate numele fisierului de intrare si de iesire care sunt date ca parametrii la rulare.

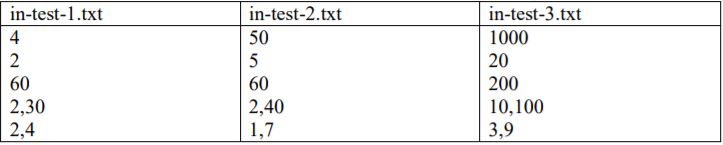
Metodele clasei SimulationManager sunt urmatoarele:

* Un constructor care primeste doua string-uri care reprezinta numele fisierelor parametrii si pe care le salveaza in variabilele corespunzatoare lor. Acest constructor intializeaza lista de clienti ca un arraylist, dupa care apeleaza functia readFromFile care, dupa cum ii sugereaza si numele citeste valorile din fisiere si le salveaza in variabilele corespunzatoare. Dupa aceea constructorul initializeaza scheduler cu parametrii: numarul de cozi si numarul de clienti si apeleaza metoda generateNRandomTasks;
* readFromFile: primeste ca parametru un string care reprezinta numele fisierului pe care il deschide si din care citeste valorile prezentate mai sus prin intermediul unui scanner. In aceasta metoda am utilizat expresii regex pentru a citi datele care erau date conform formatului exemplului din cerinta;
* generateNRandomTasks: aceasta instantiaza un obiect de tipul Random si foloseste functia nextInt() din clasa Random pentru a genera valori aleatorii, care insa sunt in intervalul specificat in fisier pentru fiecare timp, ale variabilelor unui obiect de tipul Task, instantiaza acest obiect si apoi il adauga in lista de clienti. In finalul metodei este realizata sortarea crescatoare dupa timpul de sosire a listei de clienti generatedTasks;
* metoda createFile: primeste ca parametru un string s si incearca sa creeze un fisier cu numele s, iar daca fisierul exista deja metoda va suprascrie un string gol „” in acesta;
* metoda outputInFile: are scopul de a scrie in fisierul de iesire si primeste ca parametru doua string-uri: pri,ul este numele fisierului, iar al doilea este reprezentat de string-ul care este dorit sa fie scris in fisier. Pentru aceasta metoda am folosit un printWriter;
* metoda closeTh: seteaza variabila ch cu false si apeleaza notify() pentru fiecare thread ce reprezinta o coada . Aceasta metoda este utilizata in finalul simularii pentru a inchide definitiv threadu-rile;
* run: descrie functionarea thread-ului principal. Acesta realizeaza simulare de la timpul 0 pana la timpul de simulare citit din fisier sau pana cand cozile si lista clientilor in asteptare sunt goale. Am utilizat un iterator pentru iteratia obiectelor din generatedTasks, parcurgere in care acestea sunt adaugate intr-o coada, fapt ce se realizeaza in functie de strategia descrisa in interfata Strategy, atunci cand timpul lor de sosire este egal cu timpul curent din simulare, dupa care scoate acesti clienti din generatedTasks(lista de asteptare). Aceasta metoda se ocupa si de scrierea in fisier, respectand formatul dat in cerinta, pentru aceasta ea apeleaza metoda toString din clasa Scheduler,care la randul ei apeleaza metoda cu acelasi nume din clasa Server. Dupa aceea se incrementeaza cu unu timpul curent si thread-ul este „adormit” cu ajutorul functiei sleep pentru o secunda. IF-ul din finalul buclei while este utilizat pentru afisarea timpului mediu de asteptare. Dupa bucla while, este apelata functia closeTh pentru a termina thread-urile deoarece simularea s-a incheiat;
* main: in aceasta metoda am initializat un obiect nou de tipul SimulationManager, am creat un thread nou acestui obiect si l-am pornit cu functia start().

1. Testare:

Am realizat cele trei fisiere de input din cerinta si am rulat programul avand parametrii fisierele acestea, fiecare pe rand, iar ca rezultat pentru fiecare fisier programul mi-a creeat un fisier de output corespunzator cu datele corecte.

Datele de intrare din fiecare fisier sunt urmatoarele:



Pentru cel de-al treilea fisier, numarul de clienti este 1000, iar limita superioara a timpului de procesare pe care un client il poate avea este o valoare destul de ridicata, iar timpul de simulare este de doar 200 de secunde. Din aceste cauze programul nu reuseste sa „serveasca” toti clienti din cozi pana cand timpul de simulare se incheie, o parte din clienti ramanand in acestea la finalul rularii programului.

1. Concluzii:

Dupa realizare acestui proiect, personal mi-am imbunatatit abilitatile Programarii Orientate pe Obiect si m-am familiarizat cu utilizarea threrad-urilor. Aditional, am invatat sa realizez o arhiva Java.

1. Bibliografie:

* [https://stackoverflow.com](https://stackoverflow.com/questions/10961714/how-to-properly-stop-the-thread-in-java);
* <http://tutorials.jenkov.com/java-util-concurrent/blockingqueue.html>;
* <https://blog.jetbrains.com/idea/2010/08/quickly-create-jar-artifact/>;
* <https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>;
* Mare parte din proiect a fost implementata urmand exemplul prezentat in sectiunea temei: <https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>;
* <https://howtodoinjava.com/java/multi-threading/sleep-vs-wait/>.