Simulator de cozi

Documentatie

Contiu Dragos | Grupa 30229

2021

# Cerinte functionale

* Citirea datelor de intrare prin intermediul unei interfete grafice
* Simularea lucrului cu clienti intr-un magazin
* Determinarea celei mai eficiente repartizari in functie de timpul de asteptare
* Realizarea unei statistici cu privire la timpul mediu de asteptare si momentul cel mai aglomerat
* Scrierea datelor obtinute intr-un fisier
* Folosirea threadurilor atat pentru gestionarea timpului universal cat si pentru fiecare coada in parte

# Obiective

## Obiectivul principal

Obiectivul acestei teme a fost sa implementam un program care sa simuleze lucrul cu clientii si cozile dintr-un magazin, astfel incat timpul de asteptare sa fie cat mai mic. Programul trebuie sa includa o interfata grafica cu ajutorul careia sa putem aduna date de la utilizator, cate un thread pentru fiecare coada si un thread pentru o contoriza timpul universal si a distribui uniform si eficient clientii la cozi in functie de timpul de asteptare. De asemenea, programul trebuia sa afiseze la fiecare moment de timp statusul in care se aflau cozile si clientii, intr-un fisier, iar la final trebuia afisata si o statistica legata de desfasurarea programului (timpul mediu de asteptare la cozi, momentul cel mai aglomerat de la fiecare coada).

## Obiective secundare

* Dezvoltarea de use case-uri si scenarii
* Pentru aceasta aplicatie, exista un singur use-case in care utilizatorul introduce numarul de cozi, numarul de clienti, timpul de simulare, timpul minim de sosire, timpul maxim de sosire, timpul minim de servire si timpul maxim de servire si urmareste rezultatul simularii in documentul txt pe care il genereaza programul
* Alegerea structurilor de date
* Ca sa putem efectua cat mai corect operatiile asupra clientilor trebuie aleasa cea mai buna structura de date
* Impartirea pe clase
* Dupa intelegerea cerintei, dezvoltarea use case-urilor si alegerea structurilor de date, trebuie sa gasim clasele optime pentru programul nostru
* Dezvoltarea algoritmilor
* Algoritmii alesi trebuie sa fie cei mai eficienti din punct de vedere al timpului de rulare si a memoriei aditionale folosite si trebuie adaptati situatiilor in care ne putem afla conform use case-urilor si scenariilor
* Implementarea solutiilor
* Dupa ce am realizat toti pasii de mai sus, trebuie sa implementam solutia astfel incat sa satisfacem toate cerintele analizate mai sus
* Testarea solutiei
* Dupa finalizarea implementarii solutiei, trebuie sa testam cat mai mult solutia pentru a ne asigura ca este cat mai optima posibil

# Analiza problemei

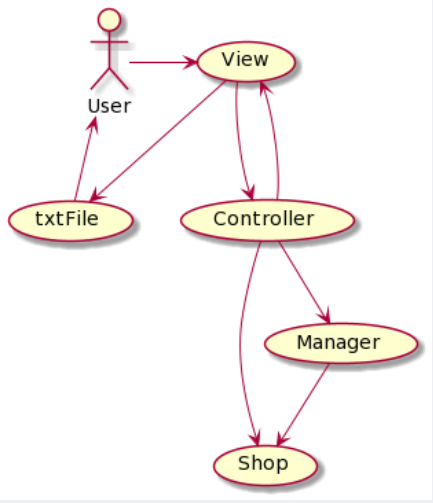


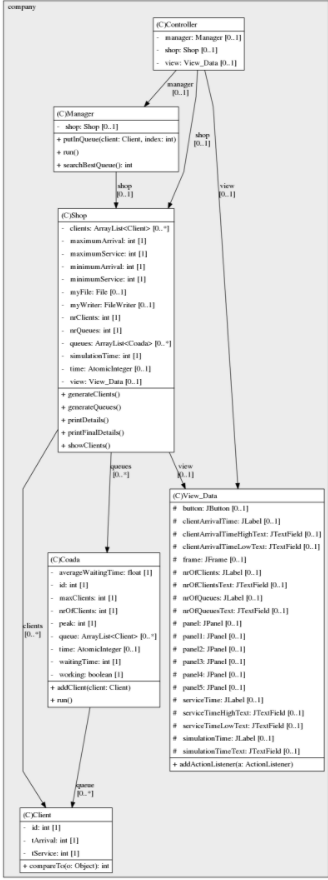
Diagrama de mai sus reprezinta singurul use-case posibil, cel in care utilizatorul introduce numarul de cozi, numarul de clienti, timpul de simulare, timpul minim de sosire al clientilor, timpul maxim de sosire, timpul minim de servire si timpul maxim de servire al clientilor. Utilizatorul poate interactiona doar cu interfata grafica care este reprezentata de entitatea „View”, iar entitatea „Controller” este liantul dintre „Shop” , „Manager” si „View”. Controller-ul transmite informatiile primite de la View mai departe la Shop si Manager, iar Shop prin metodele implementate returneaza un raspuns Controller-ului in functie de informatiile primite. In continuare, Controller-ul transmite informatiile primite de la Shop sau Manager la text file si astfel reuseste sa interactioneze utilizatorul cu programul nostru.

# Proiectare

## Structuri de date

Pentru a avea o utilizare cat mai corecta si eficenta a structurilor de date, am decis sa folosesc ArrayList pentru ca ofera functii predefinite cum ar fi clear(), add(), remove() si sort() care produc un cod mai usor de inteles si mai rapid de scris. De asemenea, elementele din ArrayList sunt de tip Client sau Coada. Clasa Client contine informatii despre clienti (id, timp de sosire si timp de servire), iar clasa Coada contine o lista de clienti si cateva date care ne ajuta la statistica si la buna functionare a programului.

* 1. **Diagrama de clase**



Clasele din diagrama alaturata sunt:

* Controller
* Manager
* View
* Client
* Shop
* Coada

Clasa Controller este liantul intre View, Manger si Shop, asigurand totodata buna functionare a programului. Clasa Manager, dupa cum ii spune si numele, se comporta ca un manager al shopului care directioneaza clientii catre casele de marcat care au cel mai putin timp de asteptare. Astfel se obtine un timp cat mai scurt de rulare al programului, iar clientii sunt multumiti fiindca asteapta putin ca sa fie serviti.

O instanta de tip Controller trebuie sa aiba cate o variabila Shop, View si Manager.

O instanta de tip Manager trebuie sa aiba o variabila de tip Shop.

* 1. **Algorimi**

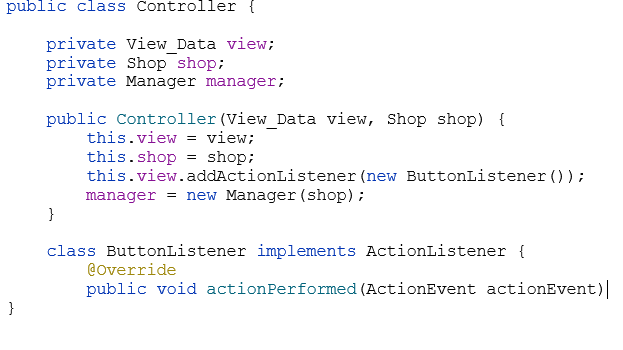
Pentru partea de algoritmi, am optat pentru folosirea unor metode implementate in clasa Collections deoarece la partea de structure de date am optat pentru ArrayList, precum:

* Sort() – care sorteaza o colectie
* Clear() – elimina toate elementele dintr-o colectie
* Add() – adauga in colectie un obiect nou dat ca parametru
* Remove() – sterge un element dintr-un ArrayList si primeste ca parametru indexul elementului pe care dorim sa il stergem

De asemenea, pentru ca tema a impus implementarea unei solutii care sa contina Threaduri, am implementat metoda run() atat in clasa Manager, cat si in clasa Coada. Pentru a reusi livrarea unui fisier text, am implementat mia multe metode try catch pentru a reusi.

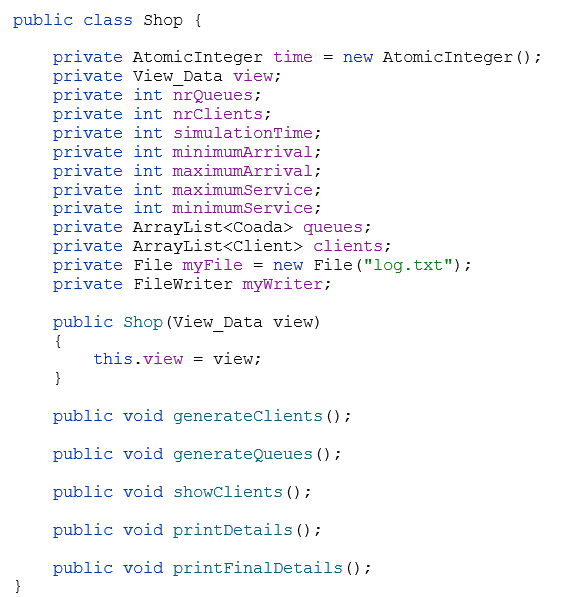
# Implementare

## Controller



Clasa Controller are nevoie de 2 paramteri in constructor (view si shop) si creeaza o instanta de tip Manager care se va ocupa de repartizarea uniforma si eficienta a clientilor care intra in magazin. In interiorul clasei Controller, am implementat clasa ButtonListener care extinde ActionListener. In momentul in care este apasat butonul de submit din interfata grafica, este pornit threadul de manager, sunt verificate datele de intrare prin structuri de tip try-catch care arunca exceptii in cazul in care sunt introduse date invalide. Se considera invalid orice input care nu poate fi gestionat de metodate „parseInt” din Integer. Controllerul este liantul dintre celelalte clase.

## Clasa Shop



Clasa Shop contine un ArrayList de cozi, adica de case de marcat, un ArrayList de clienti in care se memoreaza clientii care trebuie sa intre in magazin si pe care trebuie sa ii gestionam. File si FilWriter ne ajuta sa transferam rezultatele obtinute intr-un fisier pe care il si deschid in timpul rularii programului. Cu ajutorul variabilelor declarate in debutul clasei reusesc sa memorez datele introduse de utilizator prin intermediul interfetei grafice.

* Metoda generateClients()
* Genereaza un numar de clienti in functie de ce introduce utilizatorul. Atributele clientilor sunt generate la intamplare (random) prin intermediul clasei Random si a metodei nextInt(). Ca sa ma asigur ca sunt intre parametri introdusi de utilizator in interfata grafica, am folosit urmatoarea functie: nr = rand(high-low)+low. High reprezinta capatul mai mare al intervalului in care trebuie sa generez numarul, iar low inseamna capatul mai putin semnificativ din intervalul in care trebuie sa generez numarul. De asemenea, adaug la rezultatul obtinu valoarea lui low ca sa nu generez numere mai mici decat valoarea lui low.
* Metoda generateQueues()
* Initializeaza coada de cozi si aloca memoria necesara. De asemenea, initializeaza si timpul pentru fiecare coada.
* Motoda showClients()
* Metoda scrie in fisier detalii despre clientii pe care ii generez la intamplare la inceputul programului prin intermediul metodei generateClients(). Afisez clientii generati ca sa poate fi urmarita mai usor evolutia simulatorului de la inceput pana la final.
* Metoda printDetails()