DOCUMENTATIE

TEMA 2

NUME STUDENT: VOICU LAURA-LUISA
GRUPA: 30226

CUPRINS

1. Obiectivul temei

Obiectivul temei il constituie implementarea si proiectarea unei ce urmareste analizarea unui sistem de cozi , simuland o serie de clienti ce sosesc pentru a fi serviti si sunt impartiti strategic intr-un anumit numar de cozi, calculandu-se timpul mediu de asteptare, de servire si peak hour.

Obiectiv Secundar	Pagina
Analizarea problemei si identificarea cerintelor	3
Implementarea Claselor esentiale (Task, Service, Scheduler SimulationManagement)	7/8

Crearea unei interfete cu functionalitate multipla: permiterea introduceri datelor de catre utilizator de la tastatura si transmiterea acestora catre model (Simulation Management) si simularea in timp real a cozii	8/9
Abordarea mai multor stategii de impartire clientilor	10
Testarea functionalizatii sistemului implementat prin introducerea testelor ce variaza in numar de cozi si clienti.	

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

Descrierea problemei:

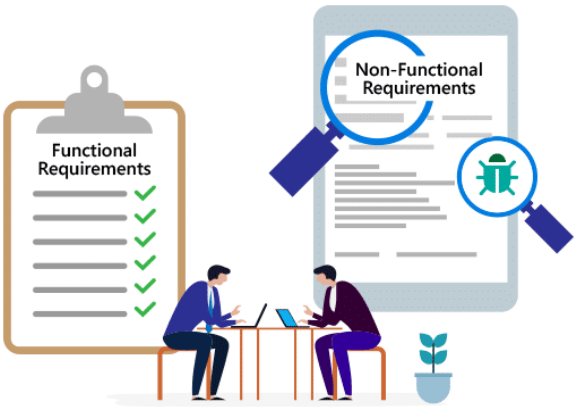
Un polinom reprezinta o expresie matematica formata dintr-o suma de termeni in care coeficientii sunt numere reale (sau complexe) iar exponentii sunt numere intregi. Termenii unui polinom poarta numele de monom , ce reprezinta un termen format dintr-o singura variabila ridicata la o putere intreaga,pozitiva.

➢ Cerinte functionale

- ***Adaugarea clientilor in coada**:*
  - *Sistemul trebuie sa permita adaugarea in coada;*
- ***Servirea clentilor:***
  - *Sistemul trebuie sa asigure servirea clientilor din coada in ordinea sosirii acestor;*
- ***Monitorizarea timpului de asteptare***
  - *Sistemul trebuie sa calculeze timpul mediu de asteptare , timpul mediu pentru service-ul fiecarui client si ora de varf.*
- ***Instiintarea clientilor:***
  - *Sistemul trebuie sa anunte clienti cand si la care dintre cozile disponibile sa se aseze , cand sunt serviti (service time-ul este consumat) si cand e nevoie ca acestia sa paraseasca coada*
- ***Gestionarea cozilor multiple:***
  - *Sistemul trebuie sa coordoneze datele, astfel incat acestea sa fie impartite corespunzator mai multor cozi.*

➢ Cerinte non-functionale

- ***Salabilitate** :*
  - *Sistemul trebuie sa poata fi scalat in mod eficient, astfel incat sa poata gestiona cresterea numarului de clienti in coada.*



❖ ***Performanta:***

○ *Sistemul trebuie sa aiba o performanta rapida si eficienta, astfel incat sa nu existe intarzieri semnificative intre adaugarea clientilor in coada si servirea acestora.*

❖ ***Flexibilitate:***

○ *Sistemul trebuie sa fie suficient de flexibil pentru a permite configurarea diferitelor politici de servire a clientilor, in functie de necesitatile specificului de afacere.*

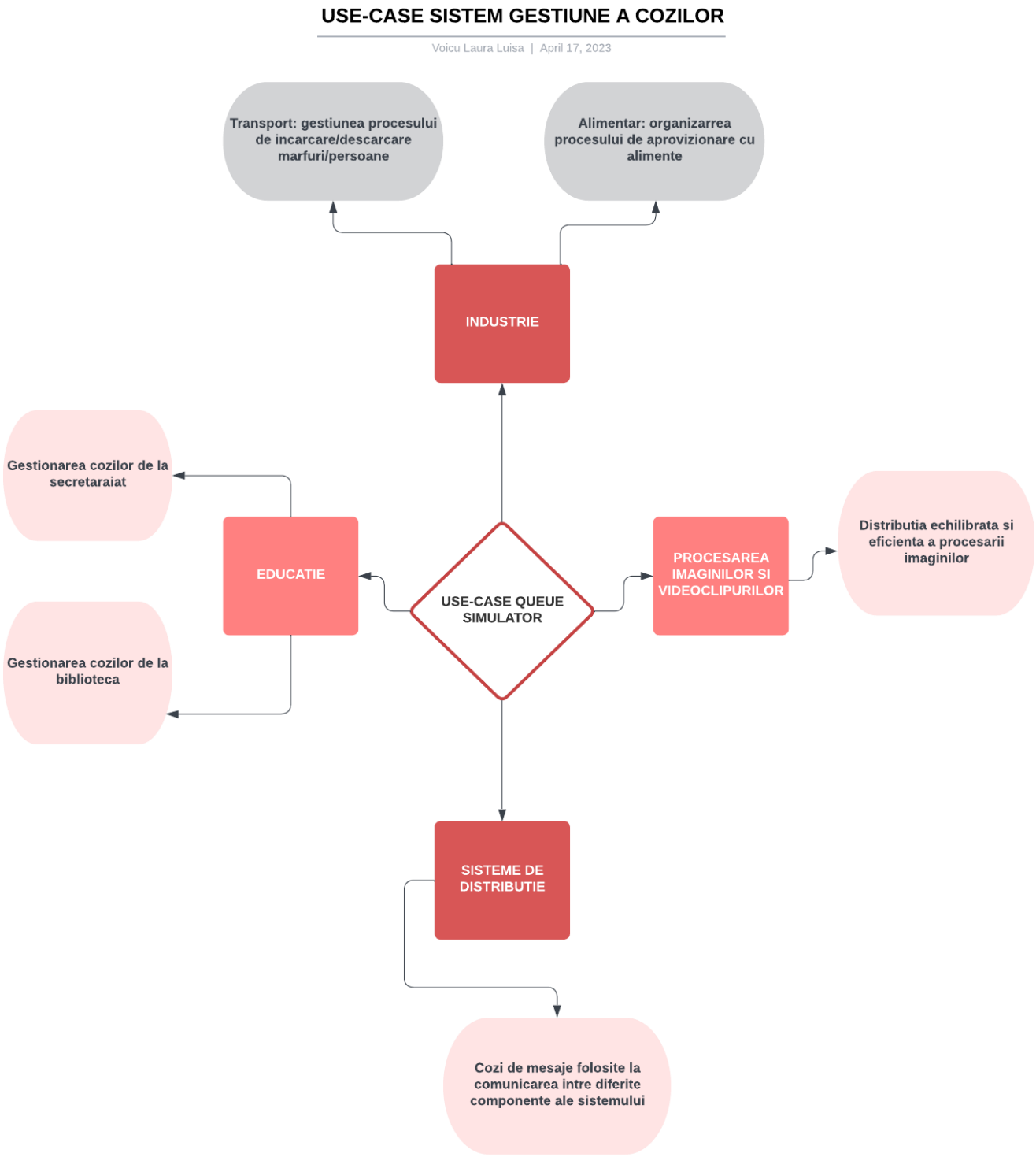
❖ ***Fiabilitate:***

- *Sistemul trebuie sa fie fiabil si sa nu permita pierderea sau coruperea datelor in timpul operatiunilor de gestiune a cozilor.*

❖ ***Usurinta de utilizare:***

- *sistemul trebuie sa fie usor de utilizat pentru clienti, astfel incat acestia sa poata intelege si sa foloseasca cu usurinta interfata utilizatorului.*

➢ USE-CASE


USE-CASE SISTEM GESTIUNE A COZILOR
Voicu Laura Luisa | April 17, 2023
Transport: gestiunea procesului de incarcare/descarcare marfuri/persoane
Alimentar: organizarrea procesului de aprovizionare cu alimente
INDUSTRIE
Gestionarea cozilor de la secretaraiat
EDUCATIE
Gestionarea cozilor de la biblioteca
USE-CASE QUEUE SIMULATOR
PROCESAREA IMAGINILOR SI VIDEOCLIPURILOR
Distributia echilibrata si eficienta a procesarii imaginilor
SISTEME DE DISTRIBUTIE
Cozi de mesaje folosite la comunicarea intre diferite componente ale sistemului

3. Proiectare

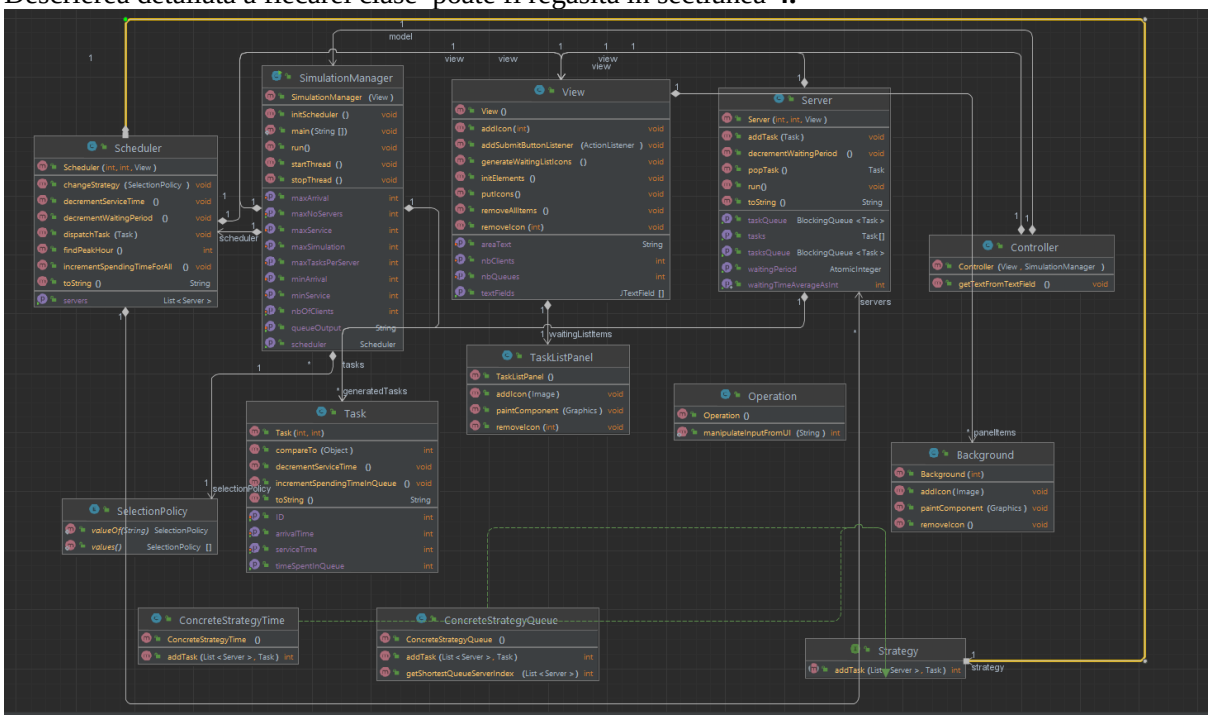
➢ Proiectarea OOP

- Functionalitatea aplicatiei este bazata pe 4 clase/interfete principale:
  - **Simulation Manager** → clasa ce simuleaza un timer; utila la monitorizarea timpului petrecut de clienti la coada;
  - **Scheduler**→ interfata ce clasifica si descrie modul in care clientii vor ocupa cozile
  - **Server** → clasa folosita pentru a simula o coada individuala
  - **Task** → clasa echivalenta cu notiunea de client → implementeaza proprietati si functionalitati specifice clientilor

  Suplimentar, aplicatia implementeaza si clasa **Operation** ce implementeaza anumite operatii ce nu au legatura directa cu operatiile legate de Queue Simulation.

De asemenea, aplicatiei este construita dupa pattern-ul MVC . Prin urmare , aceasta este alcatuita din 3 componente interconectate : **Model** (formata din clasele Monomial,Polynomial,Operation) , **View** (ce afiseaza datele din model) si **Controller** (care conecteaza modelul si view-ul si controleaza fluxul de date : primeste input-ul dat de utilizator, este translatat prin intermediul clasei Operation si transmis mai depare clasei SimulationManager si Server);

Descrierea detaliata a fiecarei clase poate fi regasita in sectiunea **4.**



- **Structurile de date** principale folosite in program sunt:
  - **Coada** de tip **BlockingQueue** ce e folosita pentru a sincroniza thread-urile corespunzatoare fiecarei cozi.
  - **List, ArrayList** folosite la controlul si organizarea componentelelor din interfata grafica.

- Printre algoritmii folositi se regasesc :

- Algoritm de separare a coeficientilor si puterilor de variabila x prin intermediul bibliotecii **REGEX**.
- Algoritmi elementari : calcul Peak Hour , calcul medie Service Time / Waiting Period, Etc.

4. Implementare

Aplicatia , construita dupa pattern-ul MVC, implementeaza cele 3 componente astfel:

- **Model :**
  - Modelul aplicatiei este constituit din mai multe clase/interfete
    - **Clasa SimulationManager** in pachetul **com.business.logic**
      - Contine campurile in care sunt retinute datele primite de la utilizator (maxSimulation, nbOfClients,maxNoServers,etc.) si campurile in care sunt calculate rezultatele cerute (average waiting time, average service time, peak hour)
      - Metodele importante implemenate in aceasta clasa sunt :
        - Generatoare de N task-uri random. In functie de datele primite de la utilizator, se vor genera random timpul necesar servirii unui client si timpul in care poate sa ajunga la una dintre cozile disponibile;
        - Metoda ce asigura functionalitatea thread-ului care calculeaza timpul. Aceasta e folosita pentru a comunica cu planificatorul si cu serverul (coada).
        - Metoda ce sorteaza timpii generati random anterior in ordine crescatoare. Aceasta apeleaza o metoda din Scheduler prin care sunt adaugati clienti in coada de asteptare corespunzatoare.

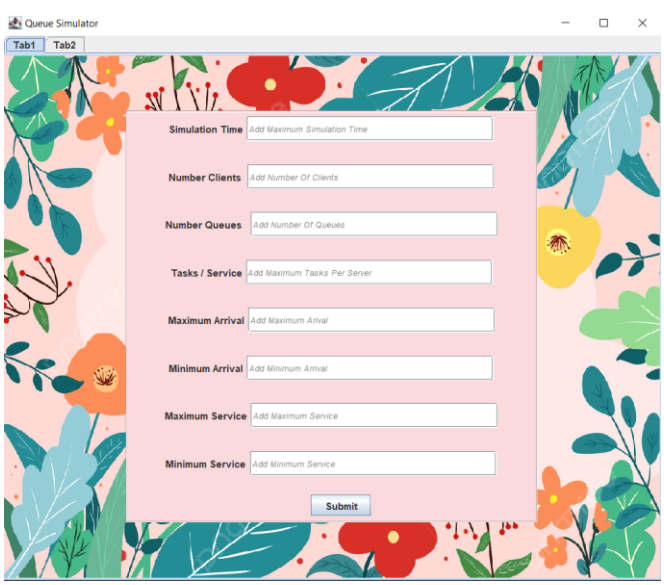
    - **Clasa Scheduler** in pachetul **com.business.logic**
      - Contine drept proprietati o lista de servere (~ cozi) , tipul de strategie aleasa, numarul maxim de cozi, numarul maxim de clienti ce pot astepta la o singura coada
      - Printre proprietatile importante ale clasei se regasesc:
        - Metoda ce schimba stategia in functie de alegerea utilizatorului (programul dispune de 2 tipuri de strategii)
        - Metoda ce comunica cu fiecare server in parte pentru a adauga task-uri/clienti
        - Metoda ce decrementeaza timpul de asteptare/ de servire al clientilor aflati intr-o coada
        - Metoda ce calculeaza ora de varf
    - **Clasa Server** in pachetul **com.model**
      - Proprietatile clasei:
        - O coada de tipul BlockingQueue ce asigura sincronizarea thread-urilor
        - Variabila de tip AtomicInteger care de asemenea garanteaza sincronizarea thread-urilor astfel incat aceasta sa poata fi modificata de un singur thread la un moment dat
      - Metodele clasei:

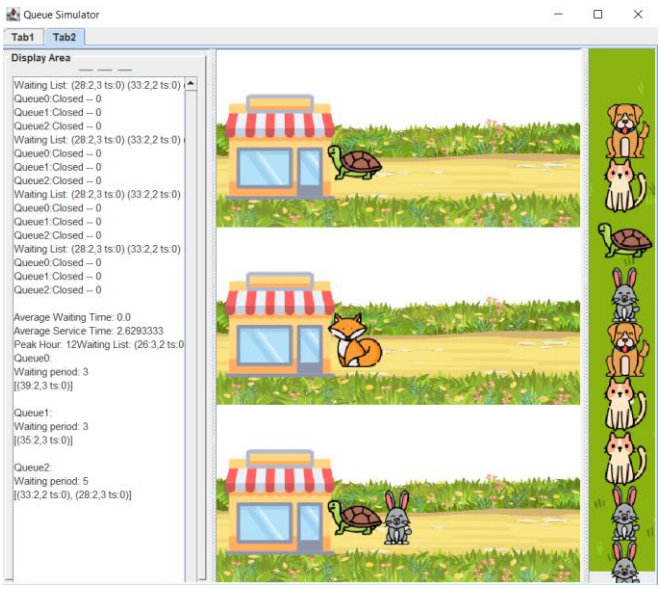
- Metoda override ce asigura functionalitatea thread-urilor mostenita din clasa Runnable
- Metode ce adauga/elimina task-uri din coada
- Metode ce decrementeaza timpul de asteptare

- **Clasa Task** in pachetul **com.model**
  - Proprietatile clasei:
    - Timpul de sosire, de servire, timpul petrecut in coada
  - Metodele clasei:
    - Metoda ce incrementeaza timpul petrecut in coada
    - Gettere,Settere
- **Clasa Operation** in pachetul **com.model**
  - Implementeaza metode ce nu au neaparat legatura cu operatile specifice simulatorului de cozi
    - Este definita o singura metoda ce converteste un string intr-un intreg

❖ **View-ul aplicatiei** (in pachetul **com.mvc**)

- In cadrul acestei clase sunt definite multiple **campuri**, clasificandu-se astfel:
  - pentru butonul de submit;
  - pentru field-urile de text;
  - pentru label-uri;
  - pentru panel-uri;
  - Icon-uri pentru simularea in timp real a cozii
  -





❖ **Interfata aplicatiei** este alcatuita din doua tab-uri:

- Primul tab contine text field-urile si label-urile corespunzatoare in care utilizatorul introduce datele necesare. Daca nu va fi introdus nimic intr-o caseta de text se va afisa mesajul “Wrong Input” si se vor folosi datele default (astfel incat sa fie mai eficienta testarea)
- Al doilea tab ilustreaza efectiv functionarea cozii in timp real;

- Tabul este impartit in 3 sectiuni:
  - Sectiunea din stanga ce contine o zona de text in care sunt afisate mesajele primite de la model (pentru a verifica functionalitatea icon-urilor ce simuleaza clientii)
  - Sectiunea din mijloc ce contine X icon-uri (shops) ; aici clientii se aseaza in linie, asteapta un timp (in functie de clientii din fata lui), dupa care ajung in dreptul shop-ului si asteapta o alta perioada de timp egala cu service time-ul clientului; cand aceasta perioada se incheie, clientul "dispare" si in locul lui vine urmatorul
  - Sectiunea din dreapta ce ilustreaza prin iconuri cu figurine generate random clientii aflati in asteptare

❖ **Controller-ul aplicatei** (in pachetul **com.mvc**) :
- Spre deosebire de restul claselor descrise pana acum, in cadrul acestei clase sunt implementate clase abstracte pentru Listeners; am abordat aceasta metoda de implementare deoarece numarul de Ascultatori este este mai natural si mai simplu de vizualizata clasa ce e apelata din controller.
- **Campuri: view** si **model**;
- **Metode:**
  - Metoda ce interpreteaza textul primit in casetele de text

5. Rezultate

Testarea functionalitatii aplicatiei a fost constituita atat din teste din perspectiva userului cu date ale caror rezultat era cunoscut, introduse direct din interfata grafica, cat si prin te

6. Concluzii

In concluzie, aplicatia ofera o modalitate eficienta si usor de utilizat pentru efectuarea operatiilor de baza cu polinoame. Proiectul in sine m-a ajutat sa ma familiarizez cu software-ul GitHub (crearea unui repository, push, commit, pull, etc) si cu IDE-ul IntelliJ IDEA. Crearea aplicatiei m-a ajutat la extinderea cunostintelor in ceea ce priveste design-ul OOP, crearea unei interfete grafice folosind Java Swing, lucrul cu expresii regulate, dar si aspecte referitoare la organizarea codului (utilizarea pachetelor, conventii de nume,etc).

Posibile imbunatatiri:

- Implementarea unor algoritmi de optimizare a performantei, astfel incat sa se poata gestiona cozi mai eficient si sa se reduca timpii de asteptare ai clientilor.

- Adaugarea unor functii de raportare si analiza, care sa permita utilizatorilor sa vizualizeze date statistice relevante, cum ar fi media de asteptare sau rata de abandon a clientilor.
- Integrarea cu alte sisteme sau aplicatii, cum ar fi programele de rezervare online sau software-ul de gestiune a stocurilor, pentru a facilita gestionarea fluxului de clienti

7. Bibliografie

I. FUNDAMENTAL PROGRAMMING TECHNIQUES – curs : https://dsrl.eu/courses/pt/
II. Programare Orientata pe Obiecte – Ionel Giosan : https://users.utcluj.ro/~igiosan/teaching_poo.html
III. https://www.flaticon.com/packs/animal-27
IV.