DOCUMENTAȚIE

Tema 2

Lihaciu Oana

Grupa 30225

CUPRINS

1. Obiectivul temei

Obiectivul principal:

În cadrul proiectului propunem dezvoltarea unei aplicații de gestionare a cozilor (cu interfață).

Obiectivele secundare:

Mai jos sunt prezentați pașii necesari pentru atingerea obiectivului principal, indicând în care capitole ale documentației urmează a fi dezvoltate:

- Analizarea problemei și identificarea cerințelor (Capitolul 2)
- Proiectarea calculatorului polinomial (Capitolul 3)
- Implementarea calculatorului polinomial (Capitolul 4)
- Testarea (Capitolul 5)

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

- **Problema**

  - principala problemă o reprezintă timpul îndelungat pe care oamenii îl petrec stând la coadă. Modul lor de repartizare nu este eficient și prin urmare se pierde timp inutil.

- **Soluția**

  - soluția reprezintă aplicația de gestionare a cozilor. Această aplicație va simula și analiza N clienți care se poziționează la Q cozi. La final de tot se obțin timpul mediu de așteptare, timpul mediu de servire și ora de vârf.

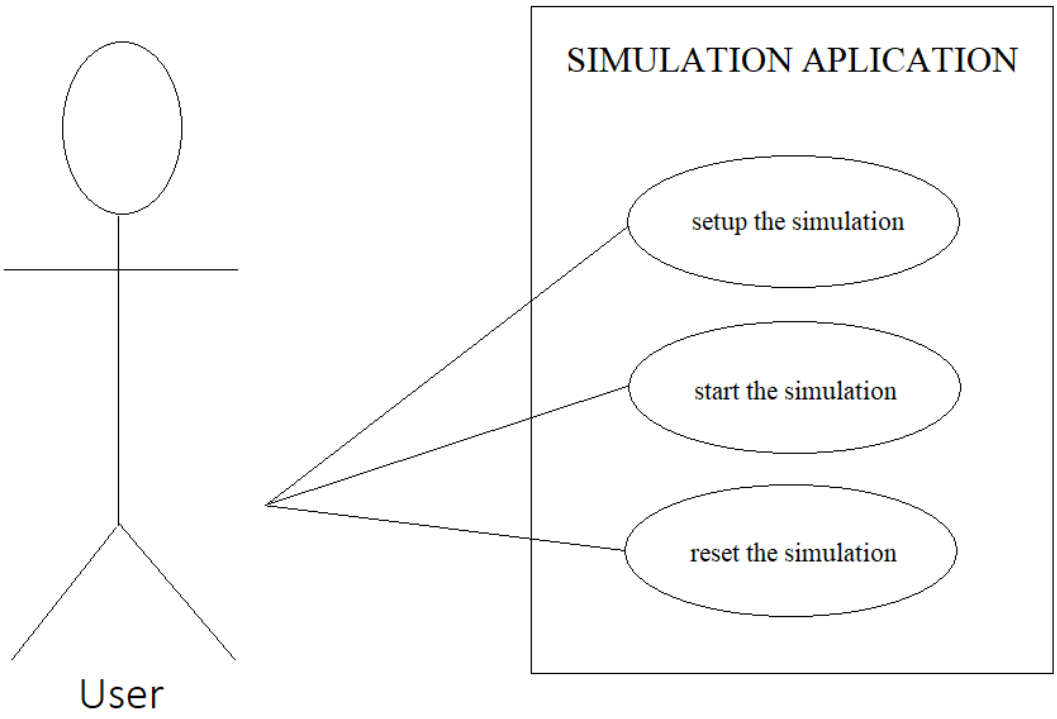
  a) Cerințe funcționale

  - Aplicația de simulare ar trebui să permită utilizatorilor să configureze simularea.
  - Aplicația de simulare ar trebui să permită utilizatorilor să înceapă simularea.
  - Aplicația de simulare ar trebui să permită utilizatorilor să reseteze simularea.
  - Aplicația de simulare ar trebui să afișeze evoluția în timp real a cozilor și a clienților care urmează să fie puși la coadă.
  - Aplicația de simulare ar trebui să afișeze evoluția în timp a cozilor într-un fișier.

  b) Cerințe non-funcționale

  - Aplicația de simulare trebuie să fie intuitiv și ușor de folosit de către utilizatori
  - Aplicația ar trebui să poată fi utilizată pe mai multe platforme și dispozitive, fiind compatibilă cu diferite sisteme de operare și configurări hardware.

- **Cazuri de utilizare**



1. **Caz de utilizare:** Configurare simulare
   Actor principal: Utilizator
   Scenariu principal de succes:

- Utilizatorul introduce valorile pentru: numărul de clienți, numărul de cozi, intervalul de simulare, timpul minim și maxim de sosire, și timpul minim și maxim de serviciu.
- Utilizatorul apasă pe butonul de validare a datelor de intrare.
- Aplicația validează datele și afișează un mesaj informând utilizatorul să înceapă simularea.

Secvență alternativă: Valori nevalide pentru parametrii de configurare

- Utilizatorul introduce valori nevalide pentru parametrii de configurare ai aplicației.
- Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă valori valide.
- Scenariul se întoarce la pasul 1.

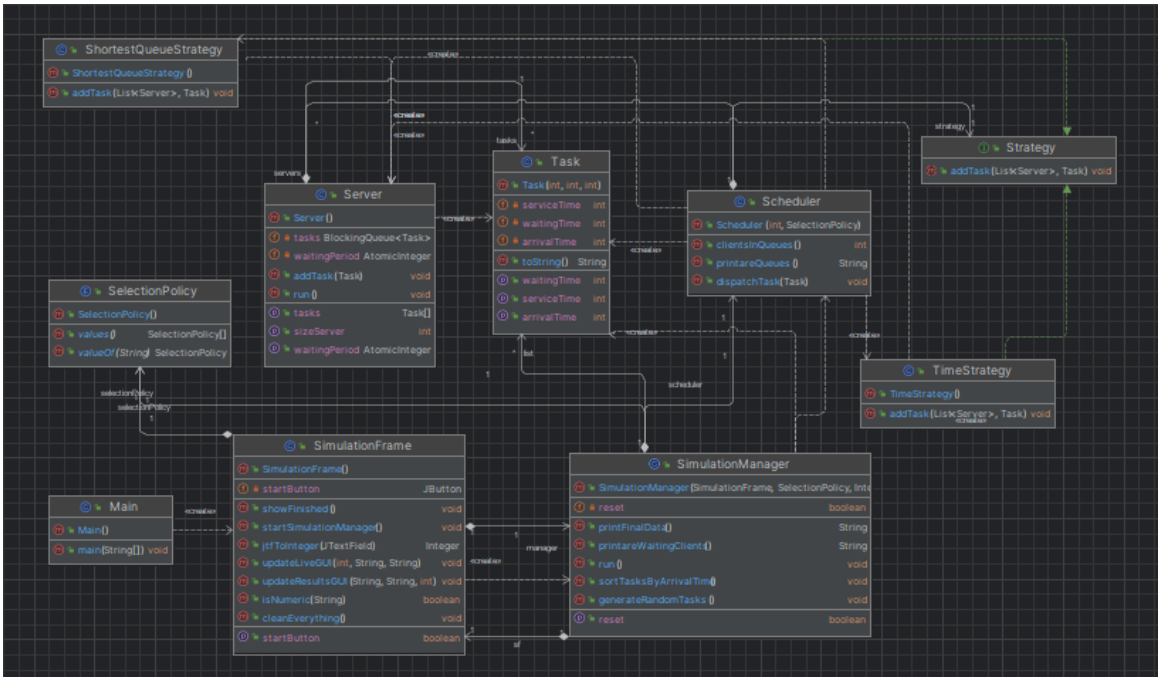
2. **Caz de utilizare:** Start simulare
   Actor principal: Utilizator
   Scenariu principal de succes:
- Utilizatorul apasă butonul de start.
- Aplicația începe simularea.

3. **Caz de utilizare:** Reset simulare
   Actor principal: Utilizator
   Scenariu principal de succes:
- Utilizatorul apasă butonul de reset.
- Setup-ul se resetează.

3. Proiectare

În faza de proiectare a aplicației noastre, am adoptat o abordare orientată pe obiecte (OOP), care ne-a permis să structurăm logic și eficient întregul sistem. Am elaborat diagramele UML de clase și de pachete pentru a vizualiza relațiile dintre entități și pentru a organiza codul într-o manieră coerentă și ușor de înțeles.

- **Diagrama UML**



- **Structurile de date**

- BlockingQueue: asigură sincronizarea și gestionarea concurenței între firele de execuție.

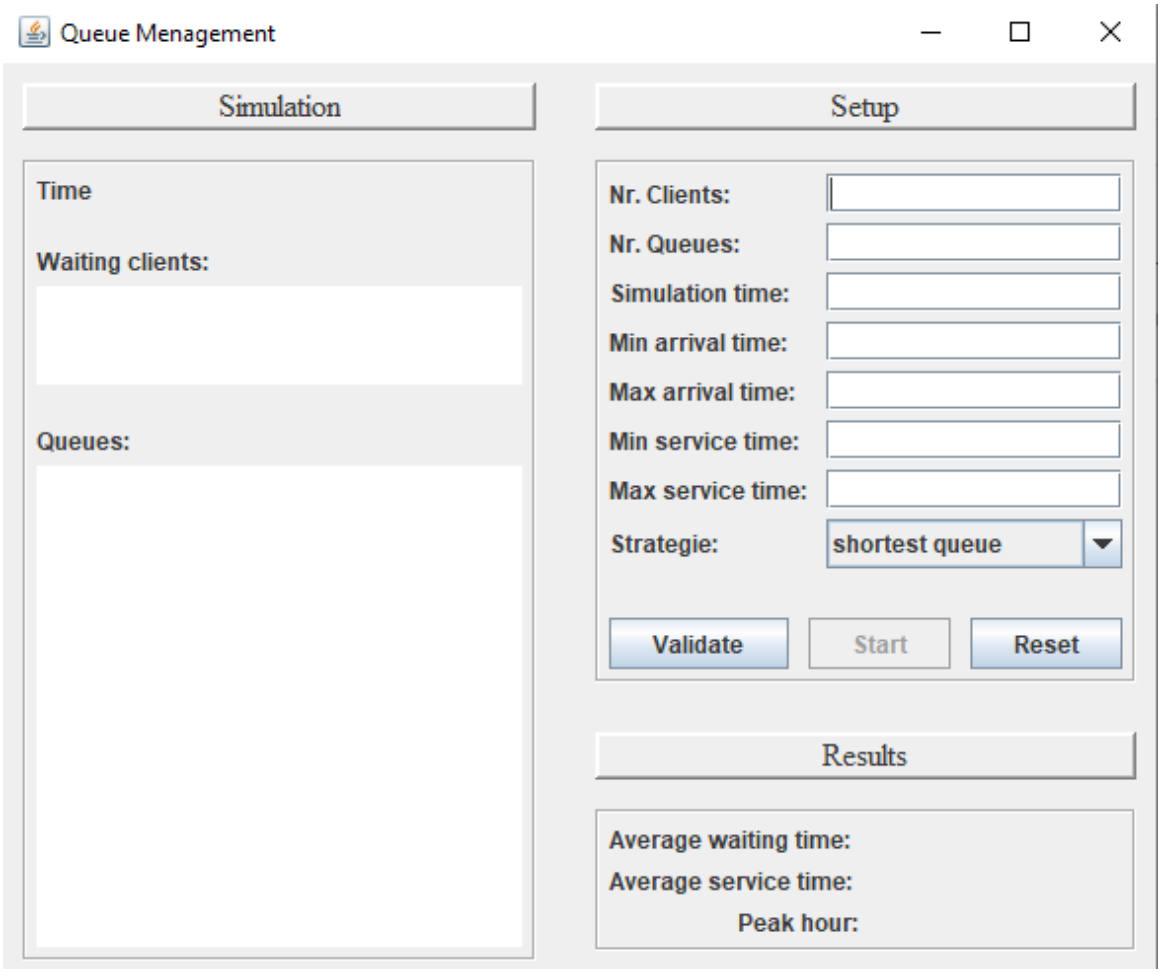
- LinkedList: pentru a stoca serverele din Scheduler și pentru a gestiona sarcinile în Server.

- String: folosit în diferite clase pentru a realiza operații de parsare și

manipulare a acestora și pentru a construi textul care este afișat în GUI.
 - **AtomicInteger:** pentru a asigura accesul concurent și atomic la timpul de așteptare al serverului în **Server**.
 - **Enum:** am definit o enumerare pentru politica de selecție a Strategy în SelectionPolicy.

- **Interfețe**
  Am folosit o singură interfață: “Strategy”, implementată de clasele “ShortestQueueStrategy” și “TimeStrategy”.

- **GUI**



4. Implementare

Clasa Task:

- **Câmpuri:**
  id: int - reprezintă id-ul clientului.
  arrivalTime: int - reprezintă timpul de sosire al clientului.
  servingTime: int - reprezintă timpul de servire al clientului.
  waitingTime: int - reprezintă timpul petrecut de client la coadă din momentul în care vine până când pleacă.

- **Metode:**
  Task(int id, int timpSosire, int timpServire): Constructor pentru inițializarea unui obiect **Task**.
  int getArrivalTime() : int: Returnează id-ul clientului.
  getServiceTime() : int: Returnează timpul de servire al clientului.

Clasa Server:

- **Câmpuri:**
  tasks: BlockingQueue<Task> - reprezintă coada cu clienți.
  waitingPeriod: AtomicInteger - reprezintă timpul de așteptare la coadă.

- **Metode:**
  Server(): Constructor pentru inițializarea unui obiect Server.
  addTask(Task newTask) : void: Adaugă un nou client în coadă și crește perioada de așteptare a cozii.
  getTasks() : Task[]: Returnează lista de cozi ca o listă înlănțuită.
  run() : void: Pornește thread-urile corespunzătoare cozilor. Elimină clienții din coadă atunci când timpul de servire ajunge la 0. Actualizează timpul de așteptare al cozii și timpul de servire al clientului.

Clasa Scheduler:

- **Câmpuri:**
  servers: List<Server> - reprezintă lista de cozi.
  strategy: Strategy - reprezintă strategia aleasă.

- **Metode:**
  Scheduler(int NoServers, SelectionPolicy selectionPolicy): Constructor pentru crearea unui Scheduler.
  printareQueues(): String: Returnează sub formă de string textul care trebuie scris la Cozi în interfață
  dispatchTask(Task t): void Apelează strategy.addTask(servers, t).
  clientsInQueues(): int: Returnează numărul total de clienți care se află la orice coadă.

Enumerația SelectionPolicy:

- **SHORTEST_QUEUE, SHORTEST_TIME**

Clasa ShortestQueueStrategy:

- **Metode:**
  addTask(List<Server> servers, Task t): Adaugă clienții în coada cu cele mai puține persoane.

Clasa SimulationManager:

- **Câmpuri:**
  Scheduler: Scheduler - reprezintă un Scheduler
  list: List<Task> - reprezintă lista de clienți care se va genera.
  SelectionPolicy: SelectionPolicy - reprezintă modul după care sunt adăugați clienții în coadă.
  fileWriter: FileWriter - reprezintă fisierul unde vor fi scrise evoluția ân timp a cozilor și rezultatele finale.
  etc

- **Metode:**
  SimulationManager(SimulationFrame sf, SelectionPolicy selectionPolicy, Integer nrClients, Integer nrQueues, Integer simulationTime,Integer minArrivalTime, Integer maxArrivalTime, Integer minServiceTime,Integer maxServiceTime): Constructor pentru crearea unui SimulationManager().
  generateRandomTasks(): void: - generează clienții împreună cu timpii lor de sosire și de servire în mod aleator.

sortTasksByArrivalTime(): void - sortează lista generată de clienți în mod crescător în funcție de timpul de sosire.
printareWaitingClients(): String: - returnează ca string clienții care încă nu au fost introduși intr-o coadă.
printFinalData(): String: - returnează sub formă de string rezultatele simulării: timpul mediu de așteptare, timpul mediu de servire și ora de vârf.
run():void - reprezintă thread-ul principal, care rulează pentru simulationTime perioade de timp. Aici se preiau clienții din lista de clienti generată aleator și se introduc în coadă. Tot aici se calculează si timpii medii de asteptare, de servire si ora de vârf și se afișează atât in interfața utilizator cât și în fișier.

Interfața Strategy:

- **Metode:**
  addTask(List<Server> servers, Task t) : void: Adaugă un client (task) în coadă (queue).

Clasa TimeStrategy:

- **Metode:**
  addTask(List<Server> servers, Task t): Adaugă clienții în cea mai scurtă coadă ca timp.

Clasa SimulationFrame (Interfața Utilizatorului):

- **Campuri:**
  Diverse componente GUI: JTextField, JButton, JLabel - reprezintă câmpurile de text, butoanele și etichetele interfeței grafice.

- **Metode:**
  Metodele care asociază acțiuni butoanelor din interfață pentru efectuarea operațiilor pe polinoame, cum ar fi **addActionListener**, **cleanEverything**, etc.

5. Rezultate

Test 1
N = 4
Q = 2
$t_{simulation}^{MAX}$ = 60 seconds
$[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 30]$
$[t_{service}^{MIN}, t_{service}^{MAX}] = [2, 4]$

- Shortest queue strategy:

```
Average waiting time:  2.25
Average service time:  2.25
            Peak hour: 9
```

- Shortest time strategy:

```
Average waiting time:  3.25
Average service time:  3.25
            Peak hour: 13
```

Test 2
N = 50
Q = 5
$t_{simulation}^{MAX}$ = 60 seconds
$[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 40]$
$[t_{service}^{MIN}, t_{service}^{MAX}] = [1, 7]$

- Shortest queue strategy:

```
Average waiting time:  4.94
Average service time:  3.98
            Peak hour: 39
```

- Shortest time strategy:

```
Average waiting time:  5.14
Average service time:  4.04
            Peak hour: 8
```

Test 3
N = 1000 Q = 20 $t^{MAX}_{simulation}$ = 200 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}]$ = [10, 100] $[t^{MIN}_{service}, t^{MAX}_{service}]$= [3, 9]

- Shortest queue strategy:

Average waiting time:	72.99
Average service time:	6.03
Peak hour:	100

- Shortest time strategy:

Average waiting time:	92.49
Average service time:	6.06
Peak hour:	100

6. Concluzii

Acest proiect m-a ajutat să înțeleg și să aplic unele concepte și tehnologii importante în programare. Am învățat cum să utilizez thread-uri și să actualizez în timp real interfața grafică GUI.

Pentru o viitoare îmbunătățire a aplicației de gestionare a cozilor, se pot lua în considerare următoarele idei:

- Extinderea aplicației pentru a suporta o gamă mai largă de politici de selecție a serverelor (cum ar fi "FIFO", "LIFO", etc.) poate oferi utilizatorilor mai multă flexibilitate în configurarea sistemului conform nevoilor lor specifice.
- Implementarea unui mecanism de gestionare a erorilor mai robust, care să ofere feedback clar și concis utilizatorilor în cazul introducerii unor date invalide.
- Adăugarea unei interfețe grafice mai interactive și intuitive, care să ofere o experiență mai plăcută și mai ușor de utilizat.
- Optimizarea algoritmilor existenți pentru a îmbunătăți performanța și eficiența aplicației.
- Extinderea documentației și adăugarea de exemple practice pentru a ajuta utilizatorii să înțeleagă mai bine modul de utilizare a aplicației și a funcționalităților acesteia.

BIBLIOGRAFIE

https://www.w3schools.com/java/java_threads.asp

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