DOCUMENTAȚIE PROIECT

Simulator de cozi

Stancu Mihai-Cristian

1. Obiectivul temei

a. Obiectiv principal

i. De a crea un simulator de cozi

b. Obiective Secundare

- Analizarea problemei și identificarea cerințelor (Capitolul 2)
- Proiectarea simulatorului de cozi (Capitolul 4)
  - descrierea tuturor pachetelor
  - descrierea tuturor claselor
- Implementarea simulatorului de cozi (Capitolul 3)
- Testarea aplicatiei a aplicației si a functionalitatii(Capitolul 5)

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

2.1 Cerințele funcționale si non-funcționale

Dupa ce am citit problema si am analizat textul, pot indentifica principalele cerinte:

1. functionale pentru calculatorul polinomial:
   - Trebuie sa am un camp mare pentru logul de evenimente in care o sa fac tranzitiile de miscare ale clientilor din waiting line in cozi;
   - Trebuie sa am 7 spinnere pentru incrementarea valorilor de cozi,client,timp de simulare,timp minim de sosire,timp maxim de sosire,timp minim de service,timp maxim de service
   - Trebuie sa am o metoda de comunicare intre cozi, trebuie sa am un observer care imi da dispatch la taskuri si trebuie sa am un mecanism de controlare a threadurilor pe parcursul intregului process de impartire a clientilor pe cozi

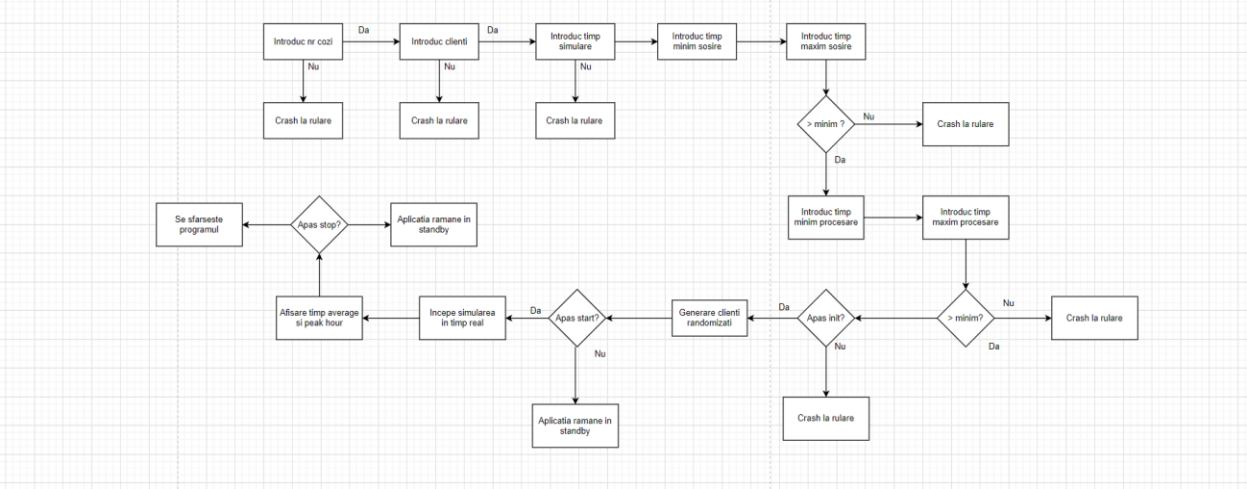
2. Cerințele non-funcționale ale calculatorului care sunt logice cand vine vorba despre ajutarea utilizatorului:
   - Trebuie sa fie intuitive
   - Trebuie sa aibe validari:
     - Sa nu fie minim arrival time mai mare decat maximul arrival time
     - Sa nu fie minim service time mai mare decat maximul service time

2.2 Descrierea diagramei de use-case

- Fata de tema trecuta, use caseurile aici sunt mai putine la numar. Aici utlizatorul poate doar incrementa spinerele de client, cozi, simulare, timp de sosire minim, timp de sosire maxim, timp de procesare minim,

timp de procesare maxim si poate controla aplicatia prin cateva butoane. Spinerele au niste valori peste care nu se poate trece, asta putand fi vazuta ca un fel de validare. Utilizatorul introduce datele de intrare conform cerintei problemei in acele spinere cu prin intermediul sagetilor, si mai apoi trebuie sa dea Init pentru crearea de butoane folosibile, adica client inscriptionati cu atibutele lor, clienti pusi in ordinea timpului de sosire. Dupa ce se da init, utilzatorul trebuie sa dea start la aplicatie, aplicatie care va simula trecerea clientilor in cozi timp de simulation time. De asemena clientul poate apasa pe clear pentru a goli ecranul sau poate apasa pe stop pentru a opri simularea. In zona de jos a programului, adica cea de rezultate, utilizatoul poate vedea secunda in care cozile sunt cele mai pline, dar si timpul mediu de asteptare al clientilor, timp mediu care se calculeaza ca suma timpilor in care clientii nu au fost primii din coada , supra numarului total de clienti. Userul poate analiza de asemenea si fisierele care contin rezultatele aplicatiei, poate observa waiting time-ul la fiecare secunda, service time-ul decrementat la fiecare secunda , continutul waiting line-ului si al cozilor la fiecare secunda din cele in care ruleaza programul pana la final.

2.3 Descrierea use-case-urilor



3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

3.1 Decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete

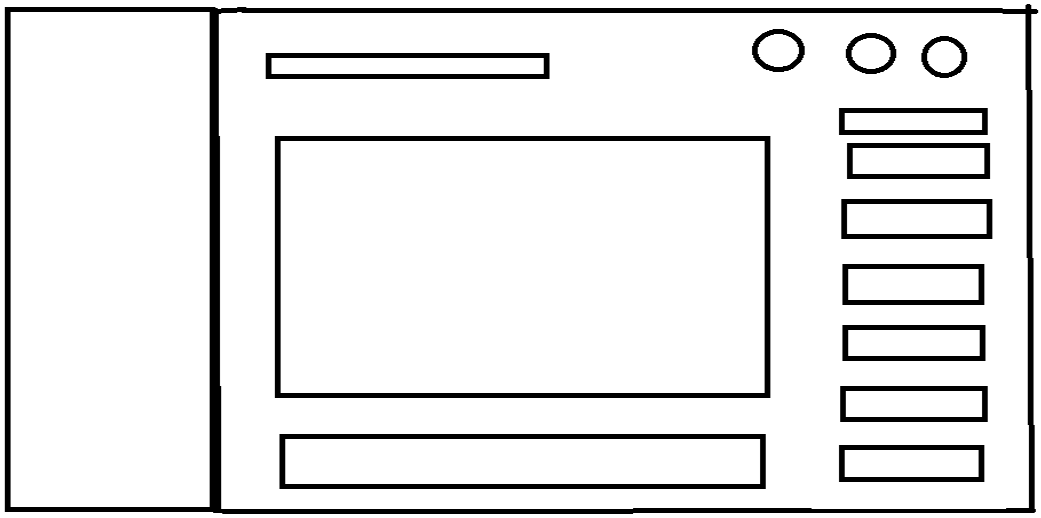
In etapa 3, dupa ce am analizat celelealte 2 etape ce parte functionala si non-functionala, mi-am creat un mock in Paint pentru a incerca sa vizualizez partea de interfata grafica si sa incerc s-o duc in arhitectura MVC pentru a avea o structura clasa si concisa pentru program.

Am 3 pachete, Model,Controller si View:

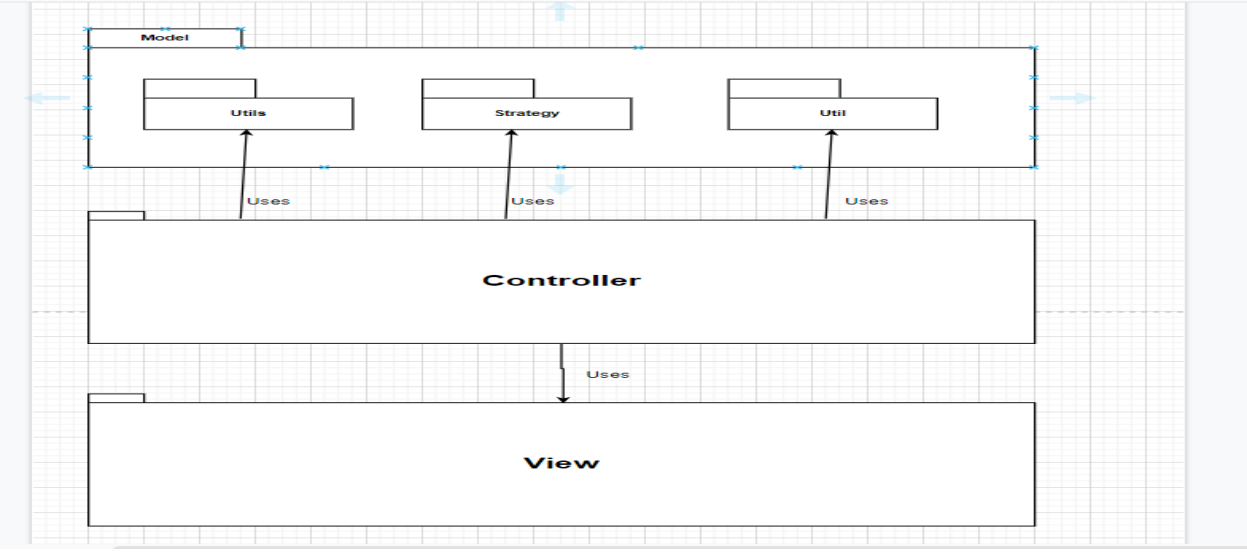
- *Controllerul* contine partea de calcul a aplicatiei si toate validarile(asta datorat arhitecturii JavaFx,care nu imi permite sa am logic ape UI in alta parte decat controller), practic toata partea de logica care s-ar gasi intr-un package de service la aplicatie pe scara mare. Am clasa PrimaryController si SecondaryController care imi managerieaza intreaga aplicatie. In partea de validari am clasa ValidateInputs care imi valideaza inputul(adica sa am maximum service time mai mare decat minimum service time maximum arrival time mai mare decat minimum arrival time) si mai am si exceptiile si utilitatile in clase aferente.

- *Viewul* contine partea de interfata grafica cu in fxml-ul primary.fxml unde am toate butoanele,paneurile si intreaga arhitectura descriptiva aplicatiei.
- *Modelul* contine clasele Pojo care s-ar mapa in database si o foarte putina bucata de logica, anume in clasa Queue(care inseamna Server) am partea de procesare a clientului si de decrementare a waitingTime-ului,iar in Scheduler doar imi initializez Threadurile si aplic metoda de dispatch pentru impartirea conform ConcreteaStrategyTime a clientilor in cozi.s

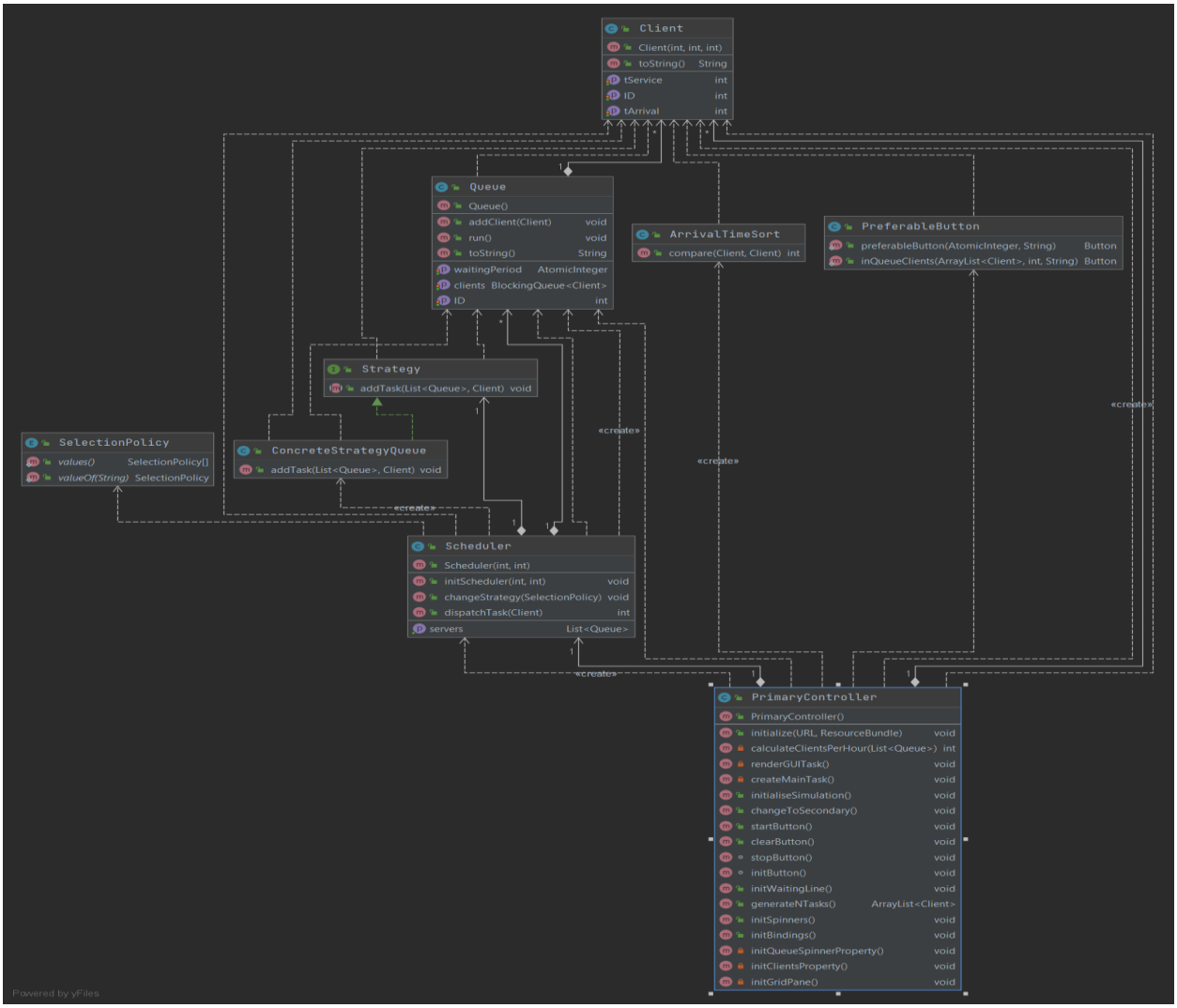
Mock



3.2 Diagrama de pachete



3.3 Diagrama de clase



4. Implementare

La implementarea temei am inceput prin analizarea a detaliilor matematice despre polinoame, si am incercat sa imi creez un sistem de date de intrare si de iesire conform cerintelor temei. Mi-am creat prima oara clasele Queue, Scheduler,si PrimaryController si de acolo am inceput sa dezvolt.

1) PACKAGE CONTROLLER

- In clasa **PrimaryController** am aproape toata logica aplicatiei. In partea de sus a clasei am componentele de FXML instantiate din primary.fxml cu ajutorul carora referintiez componentele de UI.

- Metode:
  - **InitClientsPropery**: imi fac afisarea clientilor pe UI in mod dinamic prin adaugarea unui changeListener , adica ascult apasarea butoanelor de la spinerul de client,iar la schimbarea valorii in functie de crestere sau descrestere, schimb UI conform valorilor. Asta este realizat prin intermediul IntegerPropertiurilor declarate in zona de attribute care se binduiesc cu valorea spineului. Pentru waiting line-ul la client ma folosesc de prima linie a grid paneului meu.
  - **initQueueSpinnerProperty:** imi fac afisarea clientilor pe UI in mod dinamic prin adaugarea unui changeListener , adica ascult apasarea butoanelor de la spinerul de client,iar la schimbarea valorii in functie de crestere sau descrestere, schimb UI conform valorilor. Asta este realizat prin intermediul IntegerPropertiurilor declarate in zona de attribute care se binduiesc cu valorea spineului. Pentru waiting line-ul la client ma folosesc de prima element al fiecarei linii al grid paneului meu.
  - **initBindings:** imi creez legatura dintre spiner si IntegerPropertiurile mele declarate in zona de atribute.
  - **initSpinners**: imi initalizez toate spinerele cu valorile de Factory.
  - **generateNTasks:** aici imi valoarea din ClientSpinnerProperty care este binduita cu ClientSpinner si imi creez un nr de client randomizati conform acelui numar din spinner si conform atributelor clientului de ID, ArrTime si ServiceTime .
  - **initWaitingLine:** imi iau clientii generate cu ajutor generateNTasks, si imi creez butoane din ei in momentul in care apas pe butonul de init din interfata si i pun in waiting line-ul meu de pe ui, iar cu ajutorul unui atomic integer parsez acel vector de client.
  - **stopButton:** in momentul in care apas pe butonul de stop, aplicatia se opreste.
  - **clearButton**: in momentul in care apas pe butonul de clear, spinerele mi se reseteaza la 0 si paneul de eventuri se goleste, putand sa introduc din nou valori in spinnere pentru a reincepe aplicatia.
  - **startButton:** incepe simularea prin intermediul initalizarii schedulerului cu valorile din spinnere si prin inceperea Threadului de background care imi face miscarea prin UI.
  - **changeToSecondary**: in aceasta metoda am droit initial sa ma mut pe alta scena si sa fac acolo intreaga afisare in timp real,dar nu am putut sa leg logica dintre cele doua scene asa ca am lasat acel buton jos Secondary Page care ma duce intr-o scena goala din care evident ca ma pot intoarce la primay page prin intermediul unui buton Primary Page.
  - **createMainTask**: aici imi pornesc backgroundThreadul meu prin instantierea unui Runnable in cadrul caruia am un block de try catch in care apelez metoda **renderGUITask** in care am intreaga dinamica a afisarii in timp real a clientilor prin cozi.
  - **Initialize**: metoda care apare in momentul in care implementez interfata Initializable cu ajutorul caruia pot face niste lucruri pe UI-ul meu inainte de a incepe aplicatia, precum **initSpinners, initBindings, initQueueSpinnerProperty, InitClientsPropery, createMainTask, initGridPane.**
  - **calculateClientsPerHour**: imi calculez nr de clienti intr-o secunda in toate serverele pentru a putea afisa pe TextFieldul de rezultate peak hour-ul.
  - **renderGUITask:** aici am intreaga afisare in pane a derularii in timp real a aplicatiei. Am mai multe blocuri de logica care fac diferite lucuri:
    - am logica de scriere in fisier: folosesc un BufferedWriter, si scriun secvential in timp ce parcurc metoda in zone cheie.
    - folosesc un Platform pentru dinamismul aplicatiei;

2) PACKAGE MODEL

- In clasa **ConcreteStrategyTime** am metoda addTask care este implentata datorita implementarii interfetei functionale Strategy. Aici imi pun clientul in coada cu cel mai mic waiting period, lucru pe care il fac parsand multimea de servere. De asemenea am si un index pe care il folosesc in controller pentru a stii care este indexul cozii cu cel mai mic waiting time.

- In clasa **Client** am pojoul pentru maparea clientului , care este descris de un ID, timp de sosire si timp de procesare.
- In clasa **Queue** am pojoul pentru maparea cozii, care este descrisad e un ID, o lista de client si un waiting period. Am o metoda de adaugare a clientului in coada pe care o folosesc in ConcreteStrategyTime cand calculez caoda cu timp minim si am metoda run datorata intefeti Runnable care imi ia clientul din fata cozii si pe care mi-l proceseaza conform service timeului sau, decrementand in timp real acel service timp si waiting periodul care descrie coada.
- In clasa **Scheduler** am pojoul pentru maparea observatorului. El imi trimite clientul la coada cu cel mai mic waiting time si imi da indexul catre controller pe care il folosec cand pun clientii in cozile din pane. De asemenea am logica de alegere a SelectPolicyului si tot aici imi pornesc toate threadurile cand dau init la scheduler.

3) PACKAGE VIEW

- In **fxml** am intreaga parte descriptiva a UI-ului. Am folosit un Scroll pane in care am pus un Grid pane pentru a putea da din scrollbar in jos in caz de mi se umple ecranul.In GridPane am 20 de linii si 2 coloane, pe prima coloana am cozile iar pe a doua coloana o sa am clientii. Am folosit CSS pentru stilizarea butoanelor de start, stop si clear, pentru titlu, pentru labeluri si pentru celelalte butoane. Am un progress bar care imi arata cat % din timpul de simularea trecut, am un buton pentru schimbarea scenei, lucru pe care il las pe dezvoltarea ulterioara.

5. Rezultate

- Pentru testare am 3 fisiere test:
  - In fisierul 1 am rezultate pentru inputurile:
    - Numar de clienti egal cu 4
    - Numar de cozi egal cu 2
    - Timpul de simulare egal cu 60
    - Arrival Time intre 2 si 30
    - Service Time intre 2 si 4
  - In fisierul 2 am rezultate pentru inputurile:
    - Numar de clienti egal cu 50
    - Numar de cozi egal cu 5
    - Timpul de simulare egal cu 60
    - Arrival Time intre 2 si 40
    - Service Time intre 1 si 7
  - In fisierul 1 am rezultate pentru inputurile:
    - Numar de clienti egal cu 1000
    - Numar de cozi egal cu 20
    - Timpul de simulare egal cu 200
    - Arrival Time intre 10 si 100
    - Service Time intre 3 si 9

- Pentru toate cele 3 teste, rezultatul este cel asteptat, iar valorile din fisiere respecta cerintele problemei. La fisierul 3,evident ca cozile nu s-au golit, iar un bug minor este acela ca clientii inca raman in waiting line chiar daca au intrat in coada, si dispar abia cand dispar si din coada.

6. Concluzii

1) Dezvoltari :
    a. Sa am ceva mai multe validari
    b. Imbunatarie de UI
    c. Folosirea de entitati ma avansate la Threaduri(ExecutorService de exemplu)
    d. Folosirea celei de a doua scene pentru implementarea si afisara logului de eventuri

2) Ce am invatat:
    a. am invatat JavaFx
    b. am invatat DataBinding
    c. mi-am creat o idee asupra urmatoarelor proiecte
    d. refactoring

3) Concluzii:
    a. la proiectul urmator ar trebui sa imi fac prima oara arhitectura si sa stau mai mult sa gandesc de dinainte codul, iar abia dupa ce am gandit totul, sa incep sa implementez Este mult mai rapid si este o buna practica pentru viitor, deoarece asa imi pot tine metodele mai succinte si codul mai curat.

7. Bibliografie

1) https: //www.udemy.com/course/java-se-programming/
2) https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html
3) https://www.youtube.com/watch?v=dTVVa2gfht8 (Derek Banas MVC)
4) https://www.youtube.com/watch?v=uUzRMOCBorg&t=682s (Git Tutorial)
5) https://www.youtube.com/watch?v=-NCgRD9-C6o(Derek Banas Observer Pattern)
6) https://www.youtube.com/watch?v=wiQdrH2YpT4(Derek Banas Strategy Pattern)