Simulare cozi

Documentație

Beltechi Marius Gabriel

Grupa 30227 An 2 semestrul 2

Cuprins

1.Obiectiv

Obiectivul acestui proiect a fost de proiecta si implementa un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi. Este un calculator de polinoamne care primeste ca date de intrare doua polinoame si returneaza un singur polinom, realizand operatii de adunare, scadere, inmultire si derivare.

Pe langa partea tehnica putem gasi si o interfata grafica "User Friendly" care poate fi utilizata de catre orice utilizator.

2. Studiul probelemei

Cozile sunt de obicei utilizate pentru a modela domenii din lumea reală. Obiectivul principal al unei cozi este pentru a oferi un loc pentru un „client" să aștepte înainte de a primi un „serviciu". Managementul coadă sisteme pe bază este interesat în minimizarea cantității timpul lor „clientii" sunt în așteptare în cozi înainte ca acestea sa fie servite.

Orice coadă poate fi văzută ca o pereche casă de servire - client care așteaptă la coadă, această corespondență fiind modelată după următorul aspect : fiecare coadă are clienți care trebuie procesați. Dar clientul poate să aleagă (în funcție de numărul de case de servire puse la dispoziție) cărei case de servire să fie asociat. Modul de asociere depinde de cum este văzută problema.

Acest proiect are ca rol simularea unei serii de clienti ce vor ajunge la cozi intr-un anumit interval de timp. Dupa sosire ei se vor aseza la coada unde exista cei mai putini clienti,adica au cel mai putin de asteptat, vor astepta sa le vina randul iar dupa servire vor iesi din coada. Pentru a realiza aceste lucruri, trebuie sa cunoastem momentul in care fiecare client soseste la casa si timpul de servire de care are nevoie. Despre cozi va trebui sa stim la fiecare dintre ele cat timp trebuie sa astepte noul client sosit pana cand va ajunge in varful ei, unde i se va oferi serviciul, deoarece acesta doreste sa astepte cat mai putin. Acest timp depinde de numarul de clienti de la fiecare coada si de timpul de care are nevoie fiecare in momentul in care ajunge in capul cozii.

Cerinta temei de laborator a fost proiectarea si implementarea unei aplicatii care efectiv simuleaza niste cozi cu clienti, avand in vedere mai multe aspecte care ar putea sta la baza a unei mai bune fluidizari ale acestora. Se simuleaza o serie de client care ajung la un magazine, se tine cont de plasarea acestora la niste cozi, urmarindu-se timpul in care acestia ajung la cozi,

timpul in care acestia asteapta la coada si timpul in care fiecare in parte este servit.Pentru a calcula timpul de asteptare este nevoie sa stim ora de sosire si timpul de servire. Acesti termeni sunt generati aleator pentru fiecare client in parte, astfel incat fiecare client este diferit.

Interfata grafica a aplicatiei permite utilizatorului sa introduca niste date de intrare, precum minimul si maximul timpului de sosire a clientilor, dar si minimul si maximul timpului de servire a clientilor.

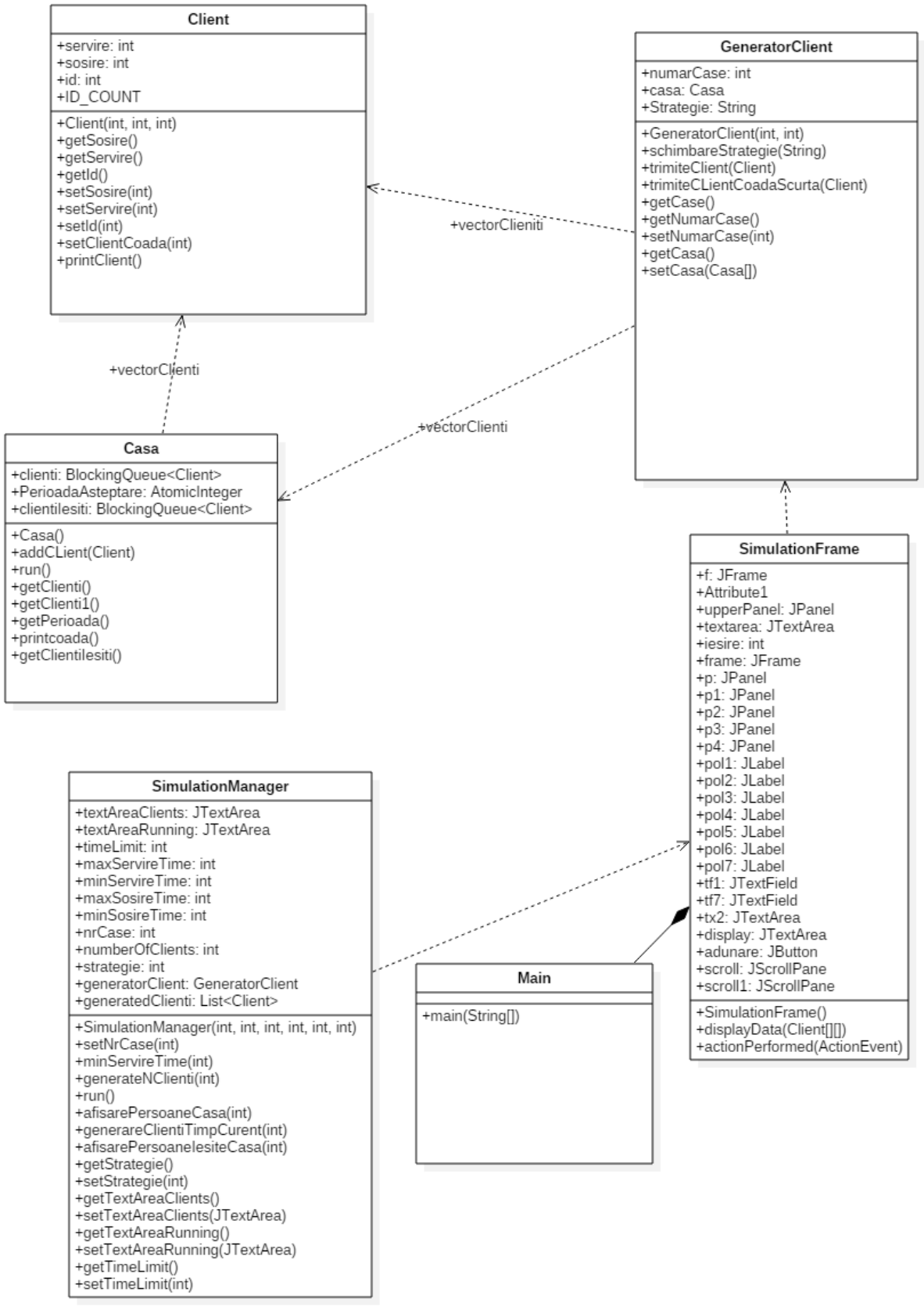
3.Implementare

3.1

Diagrame UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.

Prima versiune de UML, UML 1.0, a apărut în anul 1990 ca reacție a numeroaselor limbaje de modelare propuse pe piață. UML îi are ca fondatori pe Grady Booch, Ivar Jacobson și James Rumbaugh, așa numiții „cei trei *Amigos*”. Ei au dezvoltat limbajul bazându-se inclusiv pe limbaje de modelare deja existente, însă incomplete ca gamă de funcționalități. Printre acestea se numără și OOSE, RDD, OMT, OBA, OODA, SOMA, MOSES și OPEN/OML.


Client
+servire: int
+sosire: int
+id: int
+ID_COUNT
+Client(int, int, int)
+getSosire()
+getServire()
+getId()
+setSosire(int)
+setServire(int)
+setId(int)
+setClientCoada(int)
+printClient()
GeneratorClient
+numarCase: int
+casa: Casa
+Strategie: String
+GeneratorClient(int, int)
+schimbareStrategie(String)
+trimiteClient(Client)
+trimiteCLientCoadaScurta(Client)
+getCase()
+getNumarCase()
+setNumarCase(int)
+getCasa()
+setCasa(Casa[])
+vectorClieniti
+vectorClienti
+vectorClienti
Casa
+clienti: BlockingQueue<Client>
+PerioadaAsteptare: AtomicInteger
+clientiIesiti: BlockingQueue<Client>
+Casa()
+addCLient(Client)
+run()
+getClienti()
+getClienti1()
+getPerioada()
+printcoada()
+getClientiIesiti()
SimulationFrame
+f: JFrame
+Attribute1
+upperPanel: JPanel
+textarea: JTextArea
+iesire: int
+frame: JFrame
+p: JPanel
+p1: JPanel
+p2: JPanel
+p3: JPanel
+p4: JPanel
+pol1: JLabel
+pol2: JLabel
+pol3: JLabel
+pol4: JLabel
+pol5: JLabel
+pol6: JLabel
+pol7: JLabel
+tf1: JTextField
+tf7: JTextField
+tx2: JTextArea
+display: JTextArea
+adunare: JButton
+scroll: JScrollPane
+scroll1: JScrollPane
+SimulationFrame()
+displayData(Client[][])
+actionPerformed(ActionEvent)
SimulationManager
+textAreaClients: JTextArea
+textAreaRunning: JTextArea
+timeLimit: int
+maxServireTime: int
+minServireTime: int
+maxSosireTime: int
+minSosireTime: int
+nrCase: int
+numberOfClients: int
+strategie: int
+generatorClient: GeneratorClient
+generatedClienti: List<Client>
+SimulationManager(int, int, int, int, int, int)
+setNrCase(int)
+minServireTime(int)
+generateNClienti(int)
+run()
+afisarePersoaneCasa(int)
+generareClientiTimpCurent(int)
+afisarePersoaneIesiteCasa(int)
+getStrategie()
+setStrategie(int)
+getTextAreaClients()
+setTextAreaClients(JTextArea)
+getTextAreaRunning()
+setTextAreaRunning(JTextArea)
+getTimeLimit()
+setTimeLimit(int)
Main
+main(String[])

3.2 Clase

Clasa Interfata:

Aceasta clasa este granita dintre interfata grafica si programul propriu-zis, unde se petrece interactiunea utilizator-sistem de calcul.

Clasa Main:

Aici este clasa cu main-ul in care ii spunem programului ce sa execute. Aici doar am realizat deschiderea interfetei grafice.

Clasa Client:

In aceasta clasa am declarat clientul avand un timp de sosire,timp de servire si unul de id. Am pus Setter si Getter pe fiecare dintre acestea pentru a putea sa fie mai usor de folosit

Clasa Casa:

In aceasta clasa am declarat clientul avand o coada de clienti care au intrat, o coada pentru cei care au iesit si perioada de asteptare. Am facut un thread care imi ia clientul si asteapta o perioada de timp egala cu timpul de procesare al clientului. Si aici am folosit Setter pentru a imi returna coada de clienti.

Clasa GeneratorClient:

In aceasta clasa am declarat un vector de clase pentru a simulare cozile de la magazine. Am facut functii pentru a determina modul in care adaugam un client la coada(poate sa fie random sau la coada cu lungimea cea mai scurta) si am pus Setter si Getter pentru a seta numarul de case si pentru a returna casele.

Clasa SimulationManager:

In aceasta clasa am declarat doua JTextArea una pentru casele cu client si cealalta pentru istoricul evenimente, am declarat timpul de rulare, timpul maxim si

minim de servire si sosire pe care le-am preluat din interfata grafica, numarul de case, numarul de clienti, strategia si o lista de clienti.Am facut un thread care a rulat pana cand timpul ajungeam la timpul de rulare si acolo am afisat persoanele iesite din casa, am generat client si am afisat persoanele din casa.

3.3 Metode

Metode utilizate in clasa ***Client***

Aici am declarat structura unui client. El este format din timp sosire sit imp servire si id. Pentru fiecare dintre acestea am realizat o metoda de setare a id-ului, timp servire si sosire si una de extragere a acesteia.

```
public int getSosire() {
		return sosire;
	}

	public int getServire() {
		return servire;
	}

	public int getId() {
		return id;
	}

	public void setSosire(int sosire) {
		this.sosire = sosire;
	}

	public void setServire(int servire) {
		this.servire = servire;
	}

	public void setId(int id) {
		this.id = id;
		}
```

Pe langa acestea am mai facut o metoda toString pentru a returna id-ul.

Metode utilizate in clasa Casa

In aceasta clasa am creat o lista de client cu BlockingQueue care este o coadă care acceptă în plus operații care așteaptă ca coada să devină ne-goală la preluarea unui element și să aștepte ca spațiul să devină disponibil în coada de așteptare atunci când se stochează un element.
Metodele de blocareQueue se găsesc în patru forme, cu modalități diferite de manipulare a operațiilor care nu pot fi satisfăcute imediat, dar pot fi satisfăcute într-un anumit moment în viitor: una aruncă o excepție, a doua returnează o valoare specială (fie nulă sau falsă, operațiunea), a treia bloca firul curent pentru o perioadă nedeterminată până când operațiunea poate reuși, iar a patra blocare pentru o anumită limită de timp maximă înainte de a renunța.

Am facut un BlockingQueue pentru clientii din casa si pentru cei iesite. Pentru perioada de asteptare am folosit AtomicInteger. O operație atomică este o operație care se realizează ca o singură unitate de lucru, fără posibilitatea de interferența de la alte operațiuni. Specificatia Java garantează că citirea sau scrierea unei variabile este o operație atomică (excepția cazului în care variabila este de tip long sau double). Variabile operațiuni de tip long sau double sunt doar atomice în cazul în care a declarat cu cuvântul cheie volatile.

Am facut un thread care scoate clientul din coada il adauga in coada cu clientii iesiti si asteapta un timp egal cu timpul de servire. Am mai adaugat functii pentur adaugare in coada si pentru returnare de coada atat pentru coada clientilor iesiti cat si pentru cei de la casa si de setare a acestora.

```
public void run() {
            while (true) {
                  try {
                        Client client = clienti.take();
                        Thread.sleep(client.getServire() * 1000);
                        clientiIesiti.add(client);
                  } catch (InterruptedException e1) {
                        e1.printStackTrace();
                  }
            }

      }
```

Metode utilizate in clasa `GeneratorClient`

Metoda trimiteClient ne adauga un Client la o anumita casa random

aleasa aleator in aceasta functie.

```
public int trimiteClient(Client t) {
		int x = this.numarCase;
		Random rn = new Random();
		int answer = rn.nextInt(x);
		casa[answer].addCLient(t);
		return answer;
                                                }
```

Metoda trimiteClientCoadaScurta ne adauga un Client la o anumita casa

aleasa aleator in aceasta functie de numarul de clienti de la aceasta casa. Daca acasa are cel mai mic numar de clienti atunci se va adauga in casa.

```
public int trimiteCLientCoadaScurta(Client t) {
		int min;
		int x1 = 0;
		min = casa[0].getClienti().size();
		for (int i = 1; i < numarCase; i++) {
			if (casa[i].getClienti().size() < min) {
				min = casa[i].getClienti().size();
				x1 = i;
			}
		}
		casa[x1].addCLient(t);
		return x1;
                                                }
```

In GeneratorClient am facut un vector de case si am pornit pentru fiecare casa un thread. Restul metodelor sunt settere si gettere pentru case si numarul de case folosite in interfata grafica.

Metode utilizate in clasa `SimulationManager`

Metoda generateNClienti ne genereaza n clienti aleator si ii adauga intr-o lista de clienti pe care o vom folosi ca sa inseram la fiecare casa un client.

```
private void generateNClienti(int n) {
		for (int i = 0; i < n; i++) {
			int randomServire =
ThreadLocalRandom.current().nextInt(minServireTime, maxServireTime + 1);
			int randomSosire =
ThreadLocalRandom.current().nextInt(minSosireTime, maxSosireTime + 1);
			Client t = new Client(randomSosire, randomServire);
			generatedClienti.add(t);
		}
		generatedClienti.sort(Comparator.comparing(Client::getSosire));
	}
```

In metoda run cat timpul curent e mai mic decat timpul limita am afisat persoanele din casa si persoanele care au iesit din casa. Pentru asta am folosit doua metode separate afisarePersoaneIesiteCasa si afisarePersoaneCasa.

```
public void run() {
		int curentTime = 0;
		while (curentTime < timeLimit + 10) {
			afisarePersoaneIesiteCasa(curentTime);
			generareClientiTimpCurent(curentTime);
			afisarePersoaneCasa(curentTime);

			curentTime++;
			try {
				Thread.sleep(1000);

			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		textAreaRunning.append(String.format("Simulare finalizata la secunda
%d\n", curentTime));
	}
```

Metoda afisarePersoaneCasa are un String pe care l-am folosit sa salvez coada de clienti pentru fiecare casa si s a o printez intr-un TextArea folosind metoda SetText.

```
private void afisarePersoaneCasa(int curentTime) {
		String persoaneCasa = curentTime + "\n";
		for (int i = 0; i < nrCase; i++) {
			persoaneCasa = persoaneCasa + "Casa " + i + " " +
generatorClient.getCasa()[i].getClienti() + " \n";
		}
		textAreaClients.setText(persoaneCasa);
	}
```

Metoda afisarePersoaneIesiteCasa are un String pe care l-am folosit sa salvez coada de clienti pentru fiecare casa si s a o printez intr-un TextArea folosind metoda SetText.

```
public int trimiteCLientCoadaScurta(Client t) {
		int min;
		int x1 = 0;
		min = casa[0].getClienti().size();
		for (int i = 1; i < numarCase; i++) {
			if (casa[i].getClienti().size() < min) {
				min = casa[i].getClienti().size();
				x1 = i;
			}
		}
```

```
            casa[x1].addCLient(t);
            return x1;
        }
```

In metoda generareClientiTimpCurent am verificat care dintre cele doua strategii sa o folosesc. Daca o folosesc de exemplu pe cea random aceasta metoda ne va adauga clientul in coada daca are timpul de sosire egal cu timpul current folosind metoda trimiteClient din GeneratorClient, iar pe cea lungime mai scurta ne va adauga clientul in coada daca are timpul de sosire egal cu timpul current folosind metoda trimiteClientCoadaScurta din GeneratorClient.

3.4 GUI

Interfarta grafica sau GUI este o interfata cu utilizatorul bazata pe un sistem de afisaj ce utilizeaza elemente grafice. Interfata grafica este numit sistemul de afisaj grafic-vizual pe un ecran. Situate functional intre utilizator si dispositive electronice cum ar fi computere Pentru a prezenta toate informatiile si actiunile disponibile, un GUI ofera pictograme si indicatori vizuali, in contrast cu interfetele bazate pe text, care ofera doar nume de comenzi.

Interfata grafica are urmatoarele component: frame, panel, label,textfield, buton.

Frame este “rama” in care se adauga toate elementele de care avem nevoie pentru program.

Panel sunt cele patru panouri propriu-zise. Am facut un panou pentru primul polinom, unul pentru al doilea, altul pentru rezultatul polinomului si unul pentru operatii. In fiecare panou am adaugat label, textfield si buton representative acestuie.

Label-urile le-am folosit pentru a arata ce trebuie sa insereze utilizatorul pentru a face o anumita operatie. De exemplu, la polinom1 sa introduca un polinom in textfield-ul respective.

TextField este practice o casuta text in care utilizator introduce un text, iar programul citeste acest text si il modifica conform intructiunilor din spate. Am pus trei textField-uri, doua pentru cele doua polinoame si unul pentru rezultatul lor. De exemplu tf1 si tf2 sunt textField-urile pentru cele doua polinoame introduse de utilizator, iar tf3 este cel pentru afisarea polinomului rezultat.

Butoanele executa o anumita instructiune in momentul in care sunt apasate.

Am pus patru butoane, cate unul pentru fiecare operatie. Pentru fiecare button, s-a făcut o metodă nouă care implică ActionEvent. Astfel, de fiecare dată când are loc o acțiune la un buton, de exemplu a fost apăsat, ActionEvent-ul denumit e, transmite informația la ActionListener care așteaptă astfel de informații. Apoi, au loc evenimentele ce se afla în metodă respective.

TextField = spatii dreptunghiulare in care se pot introduce date de la tastatura. Dar pe langa asta pot fi folosite si pentru a afisa rezultatul fara a se putea introduce date de la tastatura.

\

4.**Concluzii**

In concluzie sunt de parere ca in acest proiect, am invatat sa proiectez si sa utilizez o interfata grafica, am aprofundat mai bine limbajul JAVA, implementarea paradigmelor OOP.

5.Bibliografie

1. Youtube
2. Wikipedia
3. https://beginnersbook.com/
4. https://examples.javacodegeeks.com