**Tehnici de programare fundamentale**

**Tema 2**

**Simulation application**

Pentek Tamas

Grupa: 30228

**1. Obiectivul temei**

Obiectul principal al temei este crearea unei aplicatii pentru a simula o seria de clienti care ajung la un service provider, unde fiecare client intra intr-o coada, asteapta un anumit timp, apoi primeste serviciul dorit dupa care paraseste coada. Intr-o coada sunt mai multi clienti, iar scop aplicatieo este a reduce timpul de asteptare a clientilor si a simula acest lucru. Langa aceasta, trebuie sa avem un compromis, reducand timpul de asteptare in asa fel incat sa nu crestem foarte mult costurile furnizorului de servicii. Aplicatia are o interfata grafica prin care utilizatorul paote introduce datele simularii. Dupa o simulare primim niste informatii statisitice care ne ajuta sa reducem timpul de asteptare si sa imbunatatim serviciul furnizat.

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

Aceasta aplicatie paote fi folosit in mai multe domenii, mai ales unde o companie este de fapt un furnizor de servicii. Utilizatorul aplicatie paote sa fie un angajat unei firme. Cand utilizatorul deschide aplicatia are posibilitatea de a introduce datele simularii, de exemplu durata simularii, nuamrul total de clienti, numar total de cozi, etc. Dupa aceea urmeaza rularea simularii, prin care utilizatorul urmareste „live” cum se desfasoare simularea, adica puneare clientilor in cozi si langa aceasta, utilizatorul primeste si niste feedback-uri, de exemplu: poate sa urmareasca vizual cum sunt pusi clientii in cozi.

Utilizatorul este ajutat prin mai multe guidelineuri, de exemplu: daca introduce o valoare gresita si porneste simularea apare un mesaj de eroare. Un alt feedback pentru utilizator este in momentul cand se termina simularea: apare o noua fereastra cu un text corespunzator, care indica sfarsitul simularii.

Utilizatorul poate sa faca o singura operatie, dar aceasta produce mai multe rezultate importante, este o operatia mai complexa. In ceea ce urmeaza, este prezentat operatii mai mici ca o diagrama use-case si prin descriere use-case sub forma unei liste, pe care daca utilizatorul urmareste va efectua operatia corecta si va ajunge la rezultatul dorit.

**Use Case: Urmareste vizual simularea**

**Primary Actor: utilizator**

**Main Success Scenario:**

1. Utilizatirul deschide aplicatia
2. Apare o fereastra mai mare care este impartita in 4 ferestre mai mici
3. Introduce in fereastra care are denumirea „Input” datele simularii: minimum arriving time, maximum arriving time, minimum processing time, maximum processing time, number of queues, simulation interval, si number of clients.
4. Dupa ce a introdus toate datele apasa butonul „START”
5. Utilizatorul este atent la fereastrea care are ca si titlu „Simulation”
6. Dupa o secunda porneste simularea
7. Sunt create „number of queues” cozi in care sunt introduse clientii
8. Utilizatorul poate sa verifice timpul curent al simularii care este afisat langa titilul ferestrei
9. Dupa terminarea simularii apare o fereastra noua cu un mesaj : „End of the Simulation!”
10. Utilizatorul apasa butonul „OK” dupa care este oprit programul

**Alternative Sequences:**

Utilizatorul introduce valori gresite:

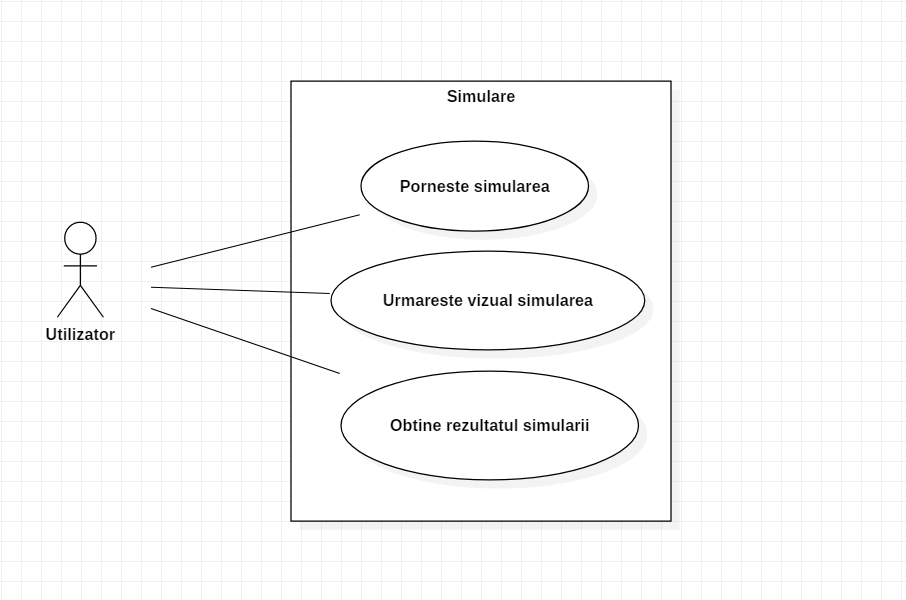
1. Utilizatorul introduce valorile dupa care apasa butonul „START”
2. Apare un mesaj de eroare: „Please insert correct values!”
3. Utilizatorul apasa butonul „OK”
4. Introduce valori corecte in in toate casutele
5. Utilizatorul continua cu punctul 4. din Main Success Scenario

**Use Case: Obtine rezultatul simularii**

**Primary Actor: utilizator**

**Main Success Scenario:**

1. Utilizatorul deschide aplicatia
2. Introduce toate valorile in casutele aflate la fereastra „Input”
3. Apasa butonul „START”
4. Dupa o secunda porneste simularea
5. Utilizatorul asteapta „Simulation interval” de secunde
6. Dupa aceea apare o fereastra cu urmatorul mesaj „End of the Simulation!”
7. Utilizatorul apasa butonul „OK”
8. Cauta fereastra care are ca si titlu „Output”
9. Citeste pe rand fiecare informatie: average waiting time, average processing time, average empty queue time si peak hour
10. Dupa aceea utilizatorul inchide aplicatia



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

Acest proiect este impartit in 4 pachete. Primul pachet este Main. Acest pachet contine o singura clasa MainClass in care gasim metoda main de unde este pornita aplicatia. Al doilea pachet este Assignment2.SimulationProject.View, in care avem 4 clase folosite pentru interfata grafica. Practic interfata grafica este impartite in 4 elemente mai importante: Input, Output, Log of Events si Simulation. Prima clasa este clas Input in are avem toate metodele si atributele care sunt folosite la fereastra de input, adica la fereastra unde utilizatorul introduce toate datele simularii si de unde este pornita simularea. A doua clasa este Output, clasa care contine toate metodele care folosim la afisarea rezultatelor dupa ce a terminat simularea. Clasa LogOfEvents este o clasa foarte importanta in aceasta clasa este descrisa fereastra care contie toate informatiile in timpul simularii. In fiecare secunda un nou mesaj este afisat in aceasta fereastra, fiecare mesaj reprezentand ori o adaugare intr-o coada, ori stergerea unui client dintr-o coada. Clasa Simulation este cea mai importanta clasa, deoarece aceasta contine simularea propriu-zisa, adica animatia de simulare. La fiecare moment se poate vedea continutul fiecarei cozi, „miscarea” clientilor, la un moment dat ajunge un client in coada, la un alt moment primul client din coada respectiva este servit si paraseste coada respectiva. Ultima clasa este MainView in care sunt legate partile interfetei grafice: cele patru ferestre mai mici, aici formeaza o fereastra mai mare, mai complexa, care contine toate datele introduse si toate rezultatele obtine dupa si in timpul simularii.

Pachetul Assignment2.SimulationProject.Model este alcatuit din 4 clase. Clasa CashRegister reprezinta o coada, implementeaza interfata Runnable si contine o lista de clienti. Aceasta clasa este folosita de un thread, fiecare instanta a clasei CashRegister este utilizat la crearea unui thread, prin care vorbim despre multithreading, adica aplicatia ruleaza pe mai multe firuri de executie. Fiindca clasa implementeaza interfata Runnable trebuie sa defineasca metoda run, in care se face stergerea clientilor din cozi. Clasa Client reprezinta un client care este introdusa intr-o coada. Fiecare client are cateva atribute importante, de exemplu arriving time, care arata timpul de ajungere; processing time, care arata timpul cat dureaza serviciul pe care clientul doreste si finish time, care este timpul, momentul cand clientul paraseste coada respectiva.

In clasa Scheduler sunt create threadurile pe baza cozilor. Aici se alege o strategie prin care se face introducerea clientilor in cozi. Exista mai multe strategii, de exemplu: strategia dupa numarul clientilor din cozi, strategia dupa timpul de asteptare din fiecare coada, strategia dupa timpul de procesare a fiecarui client. Interfata Strategy reprezinta strategia abstracta, avand o singura metoda care se face introducerea unui client intr-o coada. Clasa WaitingTimeStrategy implemeteaza interfata Strategy prin care implementeaza si metoda abstracta din interfata. In aceasta clasa se foloseste strategia urmatoare pentru a introduce clienti intr-o coada: se verifica pentru fiecare cozi timpul de asteptare. Clientul este introdus in coada in care avem un timp de asteptare cat mai mic. La fiecare client se verifica acest timp de asteptare si unde este un timp minim, acolo se introduce clientul.

Pachetul Assignment2.SimulationProject.Controller contine o singura clasa, dar aceasta reprezinta clasa cea mai importanta. Aplicatia este impartita dupa pattern-ul MVC(Model-View-Controller), cea mai importanta parte fiind Controller-ul care leaga Model-ul cu View. Clasa GeneralController din acest pachet reprezinta Controller-ul aplicatiei, unde se face legatura intre partea de model, partea de calcule si date cu interfata grafica, prin care utilizatorul ajunge la rezultatul dorit. Aceasta clasa implementeaza interfata Runnable, in metoda run se face adaugarea clientilor in diferite cozi dupa strategia respectiva. Tot aici sunt calculate rezultatele statistice a simularii si sunt afisate in fereastra respectiva, de unde utilizatorul poate sa citeasca fara probleme.

Pe schema urmatoare este prezentata diagrama UML de pachete. Din schema rezulta ca pachetul Main depinde de toate celelalte pachete, ceea ce este absolut normal, fiindca fara celelalte pachete nu putem rula aplicatia, daca rulam fara un pachet nu vom avea rezultatul dorit. Un alt lucru ce rezuta de pe figura este faptul ca pachetul Controller depinde, foloseste pachetele Assignment2.SimulationProject.Model si Assignment2.SimulationProject.View, care iarasi este un fapt absolut normal, deoarece clasa GeneralController din pachetul Assignment2.SimulationProject.Controller leaga impreuna partea de model si parte de view a aplicatiei. Pachetul Assignment2.SimulationProject.Model depinde si de view, mai exact foloseste pachetul Assignment2.SimulationProject.View ca sa afiseze rezutatele statistice calculate dupa ce se termina simularea.

Diagrama UML de pachete arata in felul urmator:

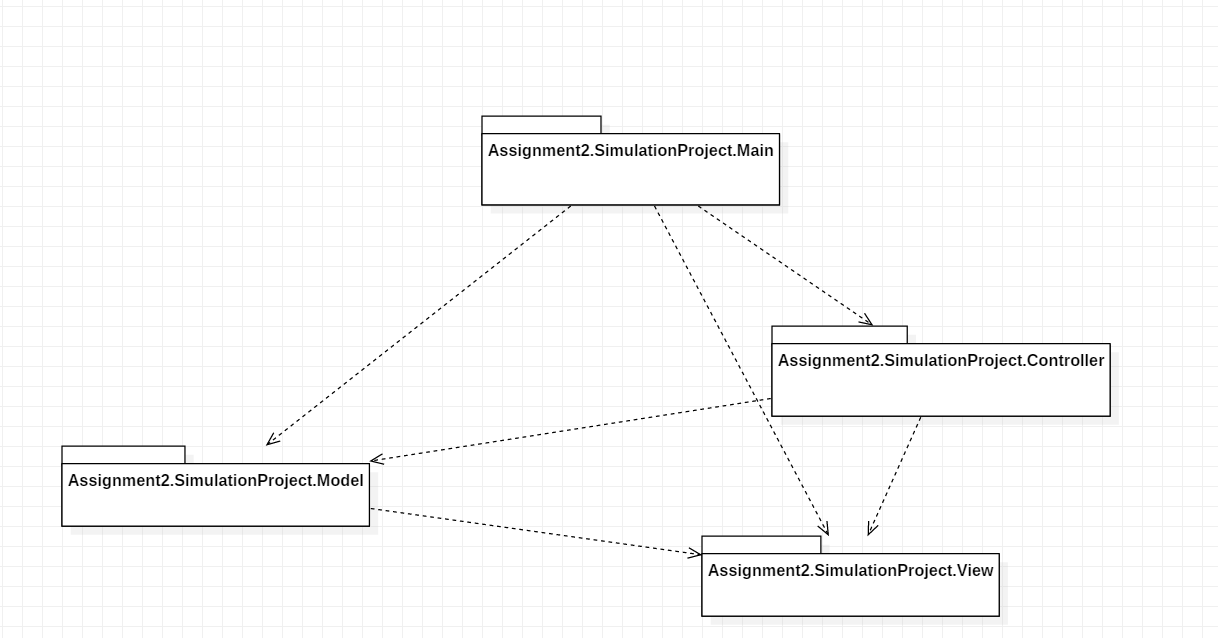
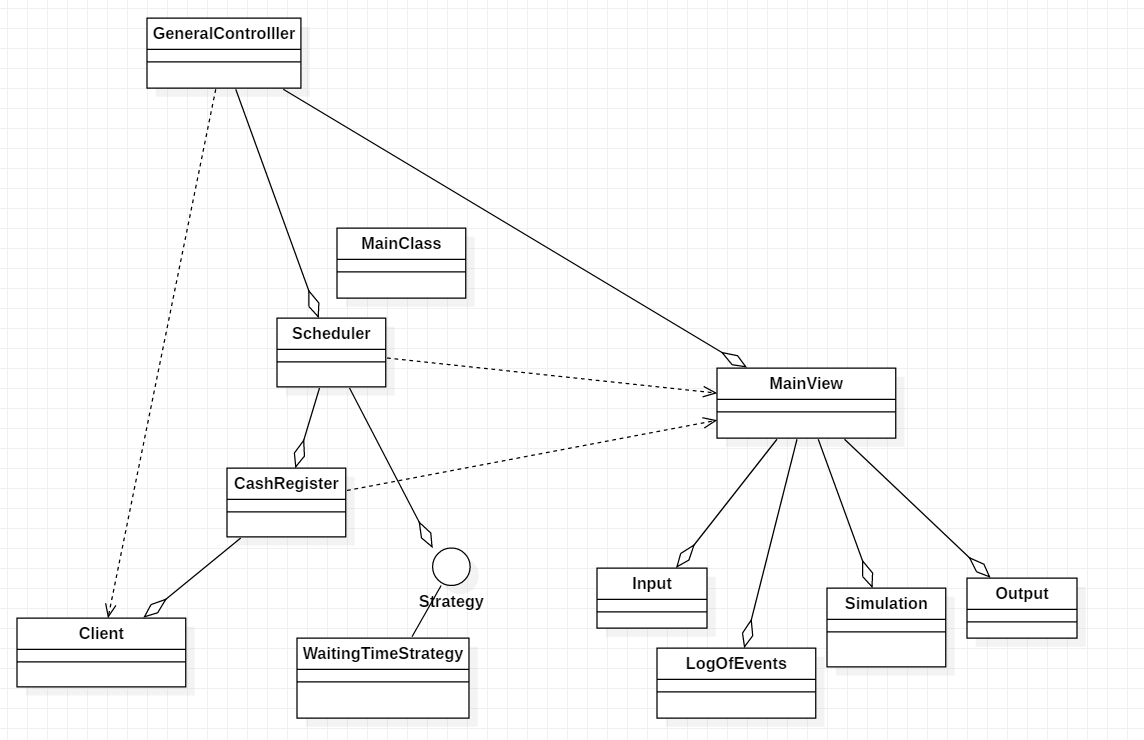


Diagrama UML de clase este prezentata pe figura urmatoare.



Clasele respecta proiectarea OOP, prin structura si prin componente, variabile si operatii folosite. Pe schema de mai sus sunt preznetate toate legaturile intre clasele. Se poate observa si de aici ca GeneralController este clasa care leagea impreuna partea de model, adica clasele Client, CashRegister, Scheduler si partea de view, adica clasa MainView.

**4. Implementare**

O metoda foarte importanta este metoda generateRandomClients din clasa GeneralController. Aceasta metoda este folosita pentru a crea nrClients de clienti. Metoda genereaza valori random pentru timpul de ajunger si timpul de procesare pentru fiecare client, apoi adauga clientul intr-o lista de clienti, din care se reia pe rand fiecare client dupa arrival time. Foarte important case face sortarea listei de clienti dupa arrival time. Codul metodei arata in felul urmator:

**public** **void** generateRandomClients(**int** nrClients) {

String name = "C ";

**for** (**int** i = 0; i < nrClients; i++) {

**int** serviceTime = **this**.minProcessingTime

+ (**int**) (Math.*random*() \* (**this**.maxProcessingTime - **this**.minProcessingTime + 1));

**int** arrivalTime = **this**.minArrivingTime

+ (**int**) (Math.*random*() \* (**this**.maxArrivingTime - **this**.minArrivingTime + 1));

Client c = **new** Client();

c.setProcessingTime(serviceTime);

c.setArrivalTime(arrivalTime);

c.setName(name + (i + 1));

generatedClients.add(c);

}

Collections.*sort*(generatedClients);

}

O alta metoda importanta este metoda run din clasa CashRegister. Metoda este implemetata pentru ca aceasta clasa imlementeaza interfata Runnable. Aceasta metoda va fi rulata de fiecare thread construit cu ajutorul unui obiect de tip CashRegister. Cele mai importante operatii facute in aceasta metoda sunt: stregerea clientului dintr-o coada, asteptarea un timp dat de timpul de procesare clientului, apoi afisarea intr-o fereastra un mesaj care arata numele clientului sters, coada din care a fost sters si momentul stergerii. Pentru lista clientilor este folosita un BlockingQueue care are niste operatii specifice, care sunt foarte utile mai ales cand folosim threaduri si lucram in paralel. Metoda take sterge primul element din coada si asteapta daca elementul respectiv nu este inca disponibila. Metoda statica sleep blocheaza threadul curent pentru un anumit timp dupa care este sters de pe grafic clientul respectiv si este afisat un mesaj in fereastra LogOfEvents.

**public** **void** run() {

**try** {

**while** (**true**) {

Client c = clients.take();

Thread.*sleep*(c.getProcessingTime() \* 1000);

**if** (**this**.interval >= c.getFinishTime()) {

**this**.view.getSim().remove(**this**.getID(), c.getName());

**this**.view.getLog().getLogs()

.append("Client: " + c.getName() + " ( Arr. " + c.getArrivalTime() + ", Pro. "

+ c.getProcessingTime() + " ) removed from Cash Register nr. " + **this**.ID + " at "

+ c.getFinishTime() + " o'clock \n");

}

waitingTime.addAndGet(-c.getProcessingTime());

}

} **catch** (InterruptedException e) {

} **catch** (Exception ex) {

System.***out***.println("EXCEPTION RUN CASH");

}

}

Simularea incepe prin introducerea datelor si prin apasarea butonului „START”. Aceasta apasara este prelucrata de un obiect de tip ActionListener. Metoda care trateaza apasarea acestui buton arata in felul urmator.

**public** **void** actionPerformed(ActionEvent e) {

**try** {

minArrivingTime = Integer.*parseInt*(view.getIn().getMinimText().getText());

maxArrivingTime = Integer.*parseInt*(view.getIn().getMaxText().getText());

minProcessingTime = Integer.*parseInt*(view.getIn().getMinServiceText().getText());

maxProcessingTime = Integer.*parseInt*(view.getIn().getMaxServiceText().getText());

nrQ = Integer.*parseInt*(view.getIn().getNrQText().getText());

simulationInterval = Integer.*parseInt*(view.getIn().getIntervalText().getText());

nrClients = Integer.*parseInt*(view.getIn().getNrClientsText().getText());

generatedClients = **new** ArrayList<Client>();

scheduler = **new** Scheduler(nrQ, nrClients, view, simulationInterval);

scheduler.selectStrategy();

generateRandomClients(nrClients);

view.updateSimulation(nrQ);

view.getSim().updateUI();

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e1) {

System.***out***.println("ERROR Starting the main THREAD");

}

t.start();

} **catch** (NumberFormatException ex) {

System.***out***.println("ERROR");

JOptionPane.*showMessageDialog*(**null**, "Please insert correct values!", "ERROR",

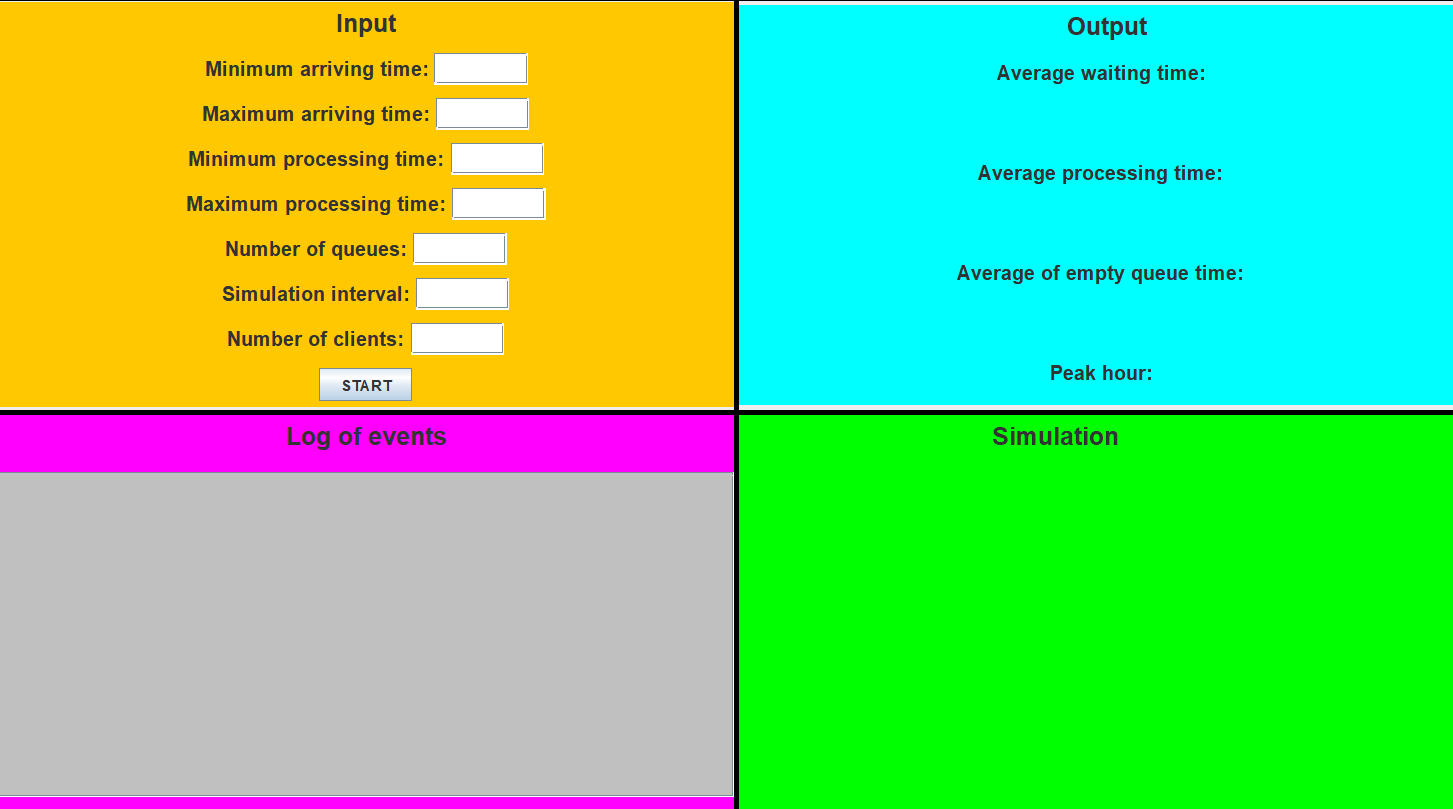
JOptionPane.***ERROR\_MESSAGE***);

}

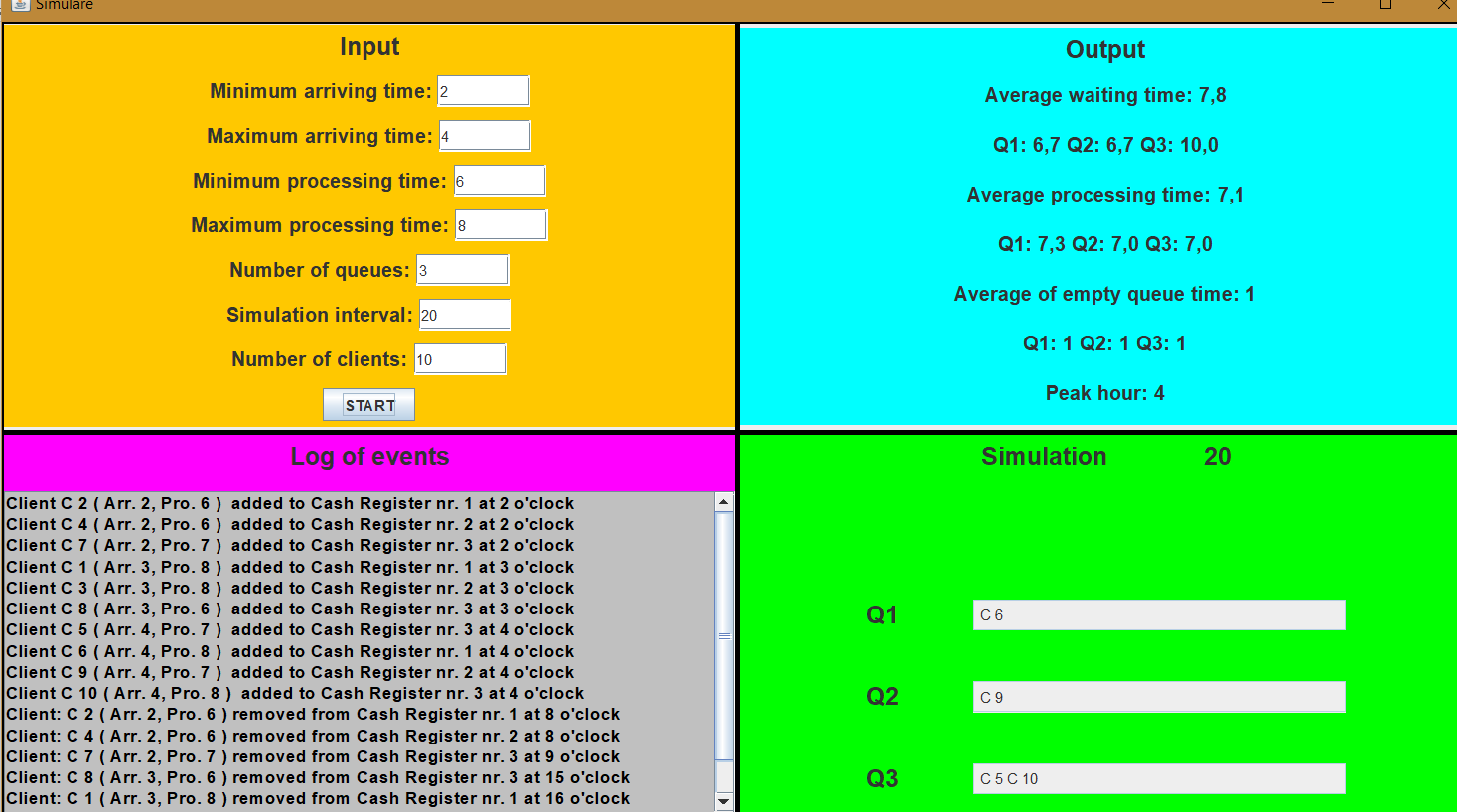
}

Prima data se face citirea datelor de la TextField-uri, dupa care valorile sunt covertite in numere intregi. Daca utilizatorul introduce o valoare gresita, de exemplu un caracter sau un text, atunci apare o exceptie de tip NumberFormatException care este tratata printr-un JOptionPane, adica se afiseaza o fereastra nou cu un mesaj de eroare: „Please insert correct values!”. Dupa citirea datelor are loc crearea unui obiect de tip Scheduler, selectarea strategiei de tip WaitingTimeStrategy dupa care sunt generate clientii. Dupa aceea este pornit threadul principal, creat dintr-un obiect de tip GeneralController, si practic se incepe simularea.

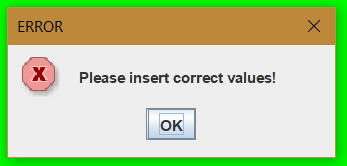
Meniul complet a aplicatiei arata in felul urmator.



Se poate observa cele 4 ferestre diferite, fiecare avand un scop diferit, aceasta fiind starea initiala a aplicatiei. Dupa rularea simularii avem toate informatiile completate, aflam toate rezultatele. Aplicatia dupa rularea simularii arata in felul urmator.



Un mesaj de eroare care apara cand utilizatorul introduce valori gresite arata in felul urmator.



**5. Concluzii**

Aceasta aplicatie este foarte folositoare, mai ales daca utilizatorii sunt persoane care lucreaza la un furnizor de servici care trebuie sa optimizeze operatiile efectuate de clienti, dar in asa fel incat sa nu aiba costuri suplimentare. Prin aceasta aplicatie este usor de simulat niste variatii, solutii din partea companiei dupa care este mai usor implementarea reala. Aceasta tema a folositoare deoarece am invatat cum se folosesc threadurile in Java, cum putem sa sincrinozam threadurile si cum putem sa proiectam o aplicatie care foloseste multithreading. Aceast aplicatie se poate dezvolta cu foarte mult, se mai pot adauga si alte operatii, de exemplu salvarea datelor statistice, inbunatatirea interfetei grafice sau generarea mai multor informatii statistice. Langa acesta putem sa extindem aplicatia ca sa monitorizam mai multe firme sau mai multe sedii a aceleiasi firma si sa introducem datele intr-o baza de date.

**6. Bibliografie**

* Pentru diagrame StarUML
* PT2018\_tema2
* https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html
* https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html
* https://docs.oracle.com/javase/tutorial/uiswing/components/border.html
* https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html