**Queue Simulation**

**TEMA 2**

Nume: Adam Cristina-Ioana

Grupa:30224

Facultatea de Automatica si Calculatoare

Specializarea: Calculatoare

Cuprins:

[1.Obiectivul temei 2](#_Toc511989091)

[2. Analiza problemei, asumptitii, scenarii, cazuri de utilizare, erori 3](#_Toc511989092)

[3. Proiectare(decizii de proiectare, diagram UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor) 8](#_Toc511989093)

[4. Implementare 10](#_Toc511989094)

[5. Testare 11](#_Toc511989095)

[6. Rezultate 11](#_Toc511989096)

[7. Cocluzii 11](#_Toc511989097)

[8.Bibliografie 11](#_Toc511989098)

# 1.Obiectivul temei

In acest proiect se dezvolta o aplicatie pentru gestiunea unor cozi in functie de timul de asteptare folosind thread-uri. Cerinta este urmatoarea:” Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients’ waiting time.”

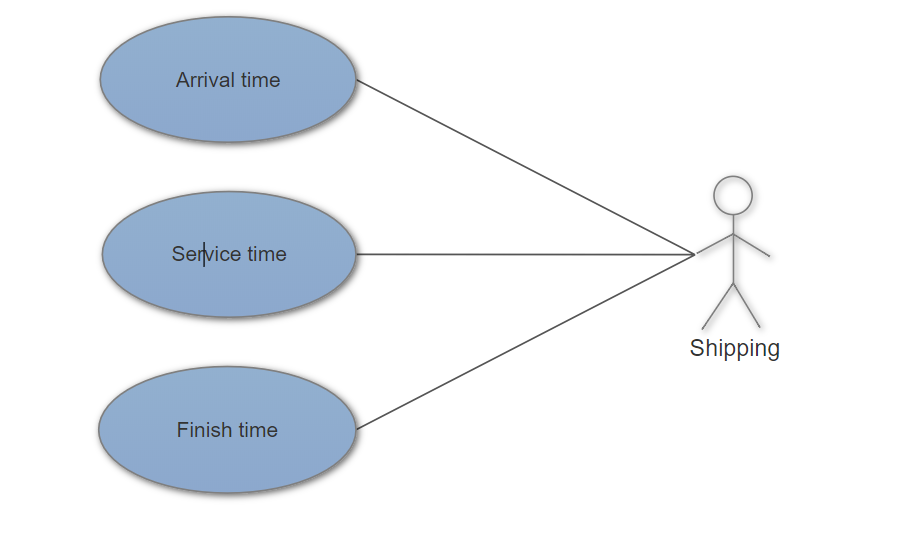
In aceasta teme se va intelege mai pe larg conceptul de cozi, dar si de thread uri. Obiectivul principal al unei cozi este de a oferi un loc pentru un client in momentul in care acesta doreste sa primeasca un serviciu. Aplicatia presupune reducerea timpului de asteptare al unui client. Acesta in momentul in care doreste sa primeasca un serviciu se pune sa astepte la coada cu timpul cel mai mic. Insa daca te gandesti in lumea reala trebuie luat in considerare si faptul ca clientul nu trebuie sa se aseze strategic doar la coada cu cei mai putini client, trebuie sa ia in considerare si cate produse au clientii din fata lui. Pentru a implimenta si acest lucru ar trebui ca pentru fiecare client sa retinem numarul de produse pe care il am. In aceasta aplicatie se va retine timpul de sosire, timpul de asteptare si timpul de finalizare al fiecarui client.

Mai pe larg se va dicuta despre aceasta tema la subpunctele urmatoare. Pentru a intelege mult mai usor functionarea se vor prezenta si cateva poze in care vom putea observa ce se intampla in aplicatie.

# 2. Analiza problemei, asumptitii, scenarii, cazuri de utilizare, erori

## 2.1 Analiza problemei

Diagrama use case ne prezinta toate operatiile pe care le poate efectua un utilizator. Utilizarea diagramelor use case nu este absolute necesara pentru a scrie o specificatie cu use case-uri, aceasta este utila pentru a crea o imagine generala asupra sistemului.

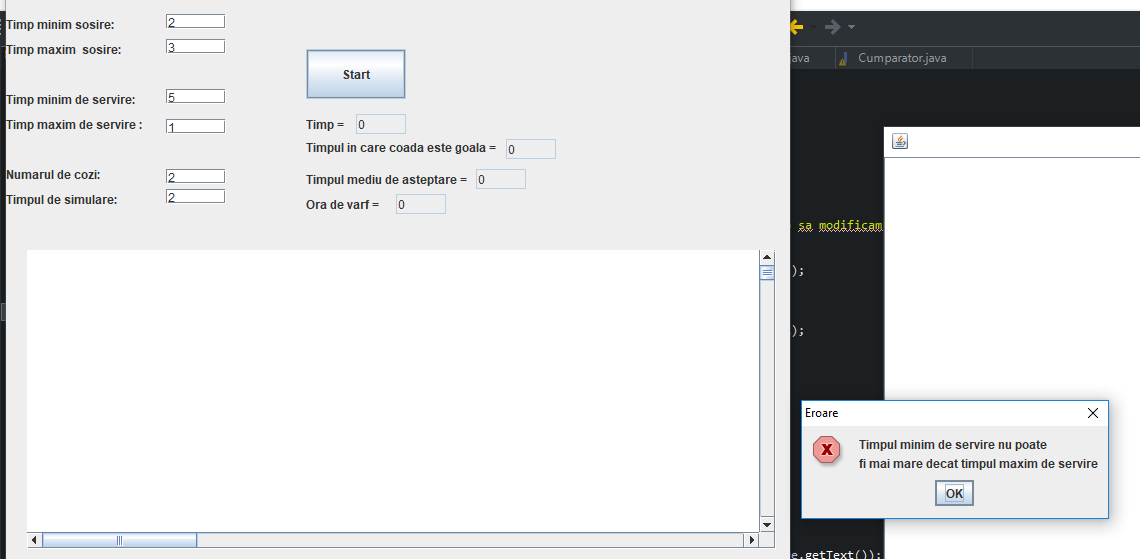


## 2.2 Asumptitii

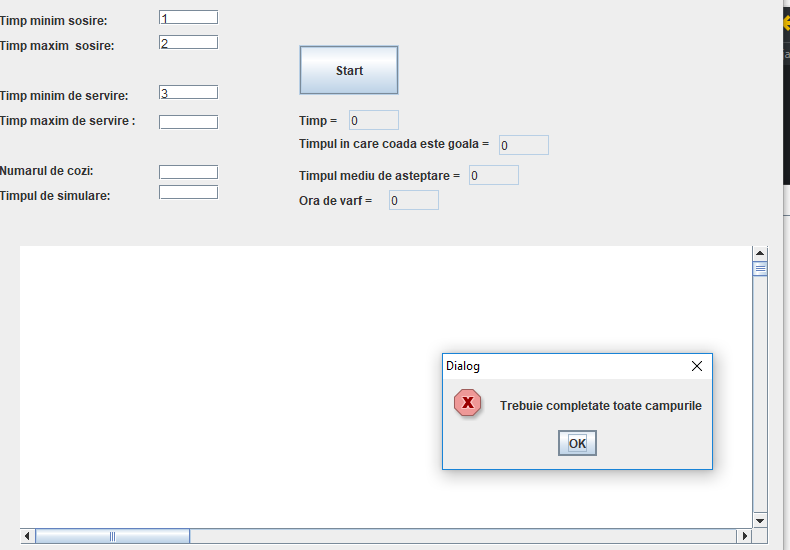
In momentul in care utilizatorul nu v- a complete toate campurile din aplicatie, ii v-a aparea un mesaj in care ii v-a spune ca trebuie sa completeze toate campurile. Un mesaje de eroare ii v-a aparea si in momentul in care introduce timpul minim de asteptare sau servire mai mare decat timpul maxim de asteptare sau servire. Desi se presupune ca este logica aceast lucru eu am specificat in interfata. Aplicatia v-a functiona atata timp cat ii v-a da utilizatorul de la tastatura.

Voi prezenta in continuare cateva etape din functionarea aplicatie, erorile care au fost luate in calcul:

1.Timpul minim de sosire sau servire este mai mare decat timpul maxim de servire sau sosire:



2. Nu sunt completate toate campurile dindin JTextField-urile din interfata:

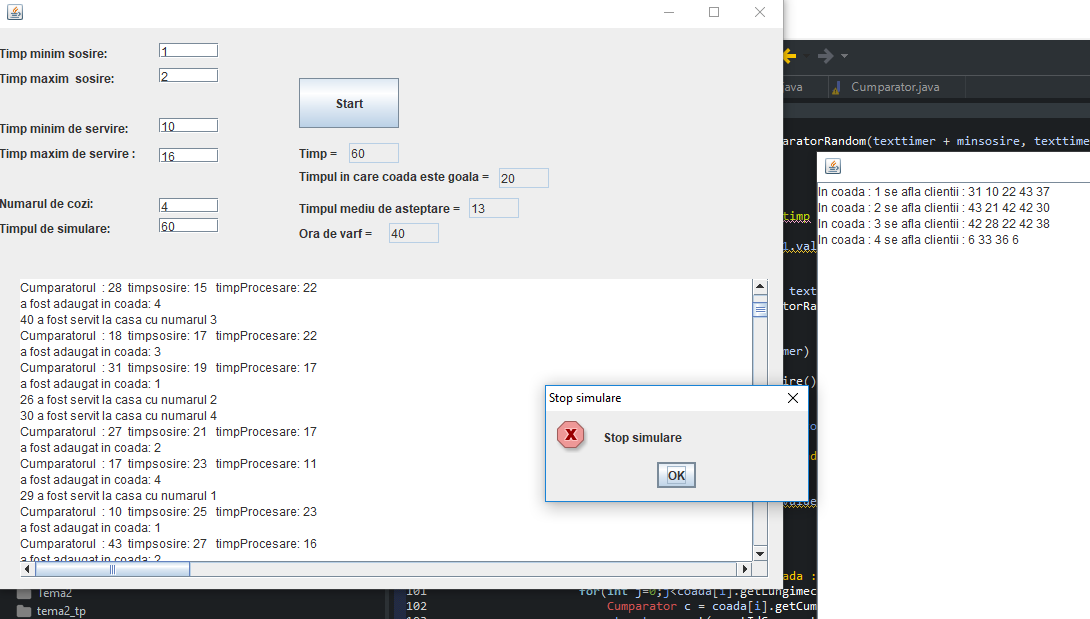
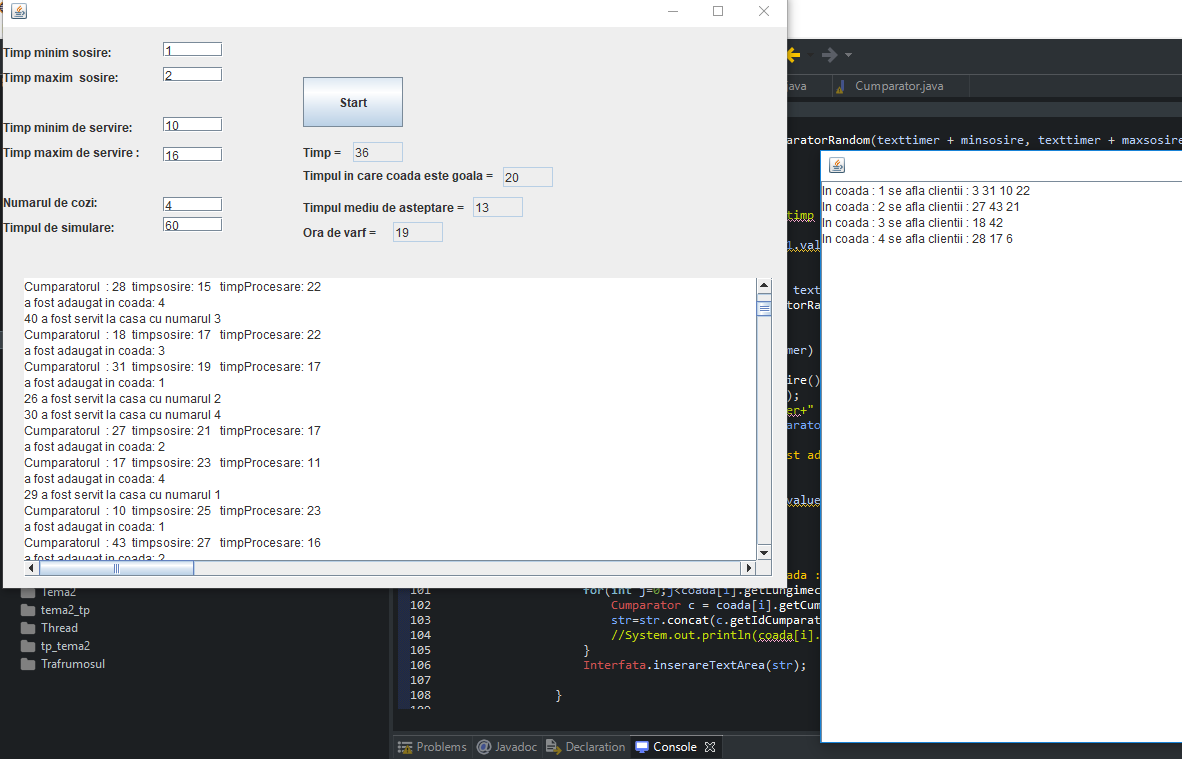


Se presupune ca utilizator nu introduce caractere special sau litere, daca se intampla acest lucru ii v-a genera un mesaj in care ii v-a scrie ca trebuie sa completeze toate campurile. Se intampla acest lucru din cauza ca eu am transmis int-uri nu sringuri in momentul in care am luat ceea ce am eu JTextField.

## 2.3. Scenarii

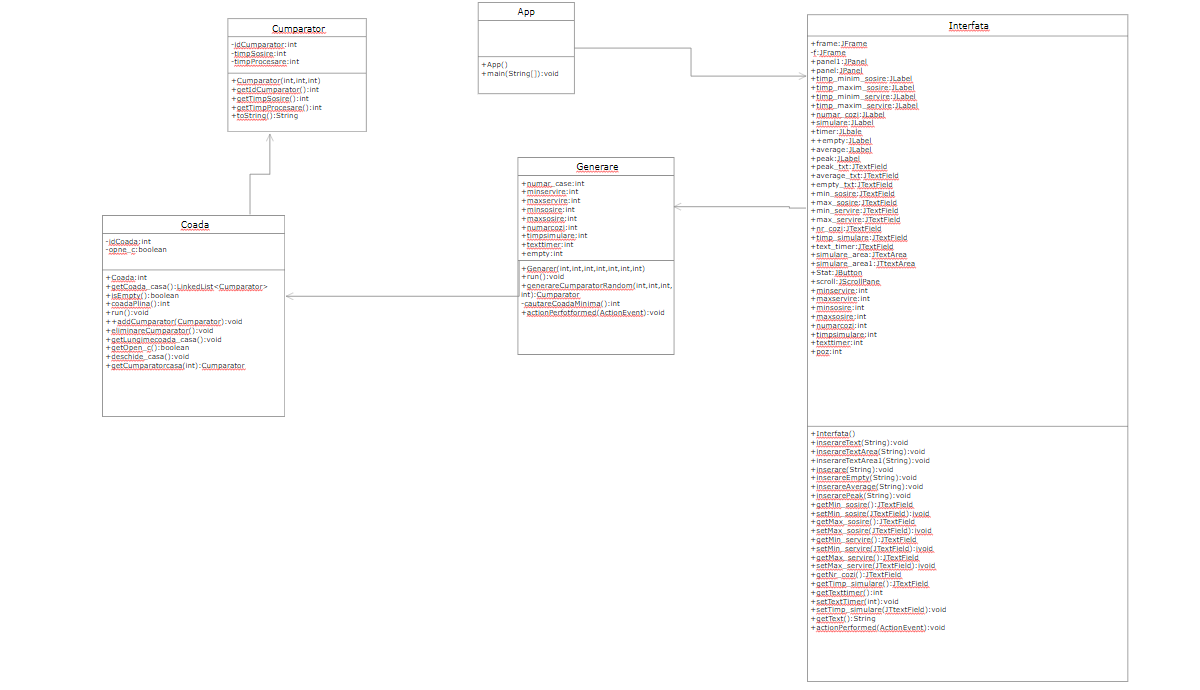
In ccontinuare se vor prezenta cateva etape din functionarea aplicatie. Functionarea aplicatie se va vede in timp real, deoarece cozile noastre de client se schimba in functie de timpul nostru de simulare introdus de la tastatura. Se considere ca se evita cazurile de mai sus pentru functionarea corecta. Atat clientii, cat si timpul de simulare sunt generate random, pentru ca nu stim cati client pot veni si ce timp sta fiecare la coada. In momentul in care imi pornesc aplicati se vor crea 2 JFrameul. In primul JFrame vom introduce noi date. Datele introduce de noi sunt: timpul minim de asteptare, timpul maxim de asteptare, timpul minim de servire, timpul maxim de servire, numurul de cozi, timpul de simulare. Aceste date le introducem noi de la tastatura. Iar rezultatul asteptat de noi v-a aparea atat in primul frame, cat si in al doilea frame.In primul textArea din primul frame imi apare timpul de asteptare si simpul de sosire al fiecarui client , cat si coada in care se afla acesta. In cel de –al doilea frame imi apare in timp real toti clientii pe care I-am generat random la coada la care se afla. In momentul in care timpul de simulare imi este egal cu timpul pe care il introduce de la tastatura imi v-a aparea un mesaj care imi v-a scrie stop simulation. Chiar daca s-a terminat timpul de simulare mie pot sa imi mai ramana clientii in coada mea.

Pentru a observa acest lucru vom pune urmatoarea poza care imi evidentiaza acest lucru.



# 3. Proiectare(decizii de proiectare, diagram UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)

O analiza mai amanuntita asupra aplicatie este prezentata de diagramele UML. UML(Unified Modelling Language) dupa cum ii spune si numele este un limbaj. Acestea sunt folosite pentru reprezentarea desfasurarii unei activitati in rularea programului. Acestea au in componenta urmatoarele elemente: numele clasei, atribute si metode. Diagrama UML pentru acest proiect este urmatoarea:



Din diagrama UML rezultata se poate observa cu usurinta ca avem 5 clasa, dintre care una este App, in care apelam clasa Interfata. Clasele sunt urmatoarele:Coada, Cumparator, Interfata si Gnerare. In clasa Interfata am creat Interfata aplicatie si am adaugat getter si setter pentru JTextField si am adaugat un ActionPerformed pentru butonul meu de start. In momentul in care utilizatorul apasa butonul de Start in anumite variable se vor converti in Integer informatiile pe care le am eu in JTextField. Si anume se va converti timpul minim de sosire, timpul maxim se sosire, timpul minim de servire, timpul maxim se desriver, numarul de cozi si timpul de simulare. Pentru a scrie informatiile in JTextArea am creat o metoda to in aceasta clasa si in acea functie am folosit functia append.

In clasa Coada am creat o lista de de cumparatori. Trebuie sa retinem pentru fiecare coada un id cu numarul ei pentru a stii in ce coada se afla clientii , urmand ca mai apoi in clasa Generare sa cautam coada cu clientii cei mai putin pentru a aduga unnou client. Clasa Coada extinde Thread. Voi explica pe scurt ce este un thread si la ce ne ajuta in aceasta aplicatie.

Conceptul de thread este folosit in programare pentru a eficientiza executia programelor, executand portiuni distincte de cod in parallel, in interiorul aceluiasi process. Cateodata insa, aceste portiuni de cod care constituie corpul threadurilor, nu sunt complet independente si in anumite momente ale executiei, se poate intampla ca un thread sa trebuiasca sa astepte executia unor instructiuni din alt thread, pentru a putea continua executia propriilor instructiuni. Aceasta tehnica prin care un thread asteapta executia altor threaduri inainte de a continua propria executie, se numeste sincronizarea threadurilor.Mai pe larg o sa puteti gasi informatii despre Threaduri in link ul atasat in bibliografie.

Multithreading apare in momentul in care se creaza mai multe cozi. Multithreading inseamna capacitatea unui program de a executa mai multe secvente de cod in acelasi timp. O astfel de secventa de cod se numeste fir de executie sau thread. Limbajul Java suporta multithreading prin clase disponibile in pachetul java.lang.

Modul de testare :se presupune ca cine foloseste aceasta aplicatie cunoaste termenul de minim si maxim, deoarece daca introducem timpul maxim mai mic decat timpul minim atata la servire , cat si la sosire ni se va genera o eroare in care ne atentioneaza ca timpul minim trebuie sa fie mai mic decat timpul maxim. Pentru a functiona in totalitate aplicatia trebuie introduse informatii in fiecare casuta, si trebuie evitata introducerea de alte caractere, deoarece eu in momentul in care imi iau informatii din JTextField il convertesc in int.

Modul de tratare a erorilor: apar mesaje de eroare de cate ori intoduc ceva gresit sau nu folosesc corect aplicatia. Am folosit try si catch pentru tratarea acestor erori.

In clasa Generare se creaza o clasa Generare care extinde thread. In aceasta clasa metodei Generare i se transmit ca parametrii timpii pe care ii avem in interfata, numarul de zozi si timpul de simulare.Initial acest timp sunt initializati la 0 dupa care in momentul in care pornesc metoda run acestia se schimba cu ceea ce am eu in interfata. In aceasta clasa avem o metoda care ne genereaza un client random ,dar si timpul in care acesta soseste la coda si timpul in care acestuia i se proceseaza comanda. Dupa ce adaug un element in coada eu trebuie sa caut coada cu cei mai putini client si in acea coada sa adaug urmatorul client si acest process se tot repeat pana in momentul in care se termina timpul de simulare. Terminarea timpului de simulare nu inseamna neaparat ca cozile mele trebuie sa ajunga sa fie goala, deoare daca e e foarte aglomerat nu o sa se termine in acel timp toata coada.

# 4. Implementare

Dupa cum am specificat si mai sus eu am 5 clase in acest proiect. Aceste clase sunt adaugat in pachete diferite, deoarece am respectat pasii pentru creerea unui proiect care sa fie Model-View –Controller.MVC se foloseste pentru o ordonare mai buna a claselor si pentru a stii exact ce se intapla in fiecare pachet.

In model se afla clasele:Coada si Cumparator. Aceasta parte a controlatorului manipuleaza operatiunile logice si de utilizare de informative(trimisa dinainte de catre rangul sau superior) pentru a rezulta o forma usor de inteles.

In pachetul View se afla clasa Interfata. Acestui memeebru al familiei ii corespunde reprezentarea grafica, sau mai bine zis, exprimarea ultimei forme a datelor: interfata grafica interactioneaza cu utilizatorul final. Rolul sau este de a evidentia informatia obtinuta pan ace ea ajunge la Controller.

In pachetul controller se afla clasa Generare. Cu acest element putem controla accesul la aplicatia noastra. Iar in pachetul DemoProject se afla App, in care e apleata clasa Interfata si in care se afla clasa main bineinteles.

Pentru a nu primi mesaje de eroare voi specifica in continuare modurile de utilizare ale aplicatiei:

1. Un pas important pentru a nu genera eroare este acela de a complete toate JTextField urile din Interfata. Daca nu se intampla acest lucru v-a aparea un mesaj in care v-a scrie “Toate campurile trebuie completate”.
2. Un alt pas importat este acela de a introduce un timp minim de servire mai mic decat timpul maxim de servire, dar si un timp maxim de servire mai mic decat timpul maxim de servire. Daca se introduce gresit v-a aparea un mesaj de eroare prin care te atentioneaza ca nu ai introdus correct acesti timp.
3. Pentru a putea observa evolutia cozilor in timp real se recomanda ca timpii de procesare sa fie mai mari decat timpii de servire. Deoarece in momentul in care mie imi tot vin client si timpul de procesare al clientului este foarte mare clientul respective v-a sta mult la coada.
4. Se presupune ca utilizatorul v-a introduce numai variabile de tip integer in aceasta aplicatie. Verificarea daca un utilizator a intodus integer sau string nu a fost facuta, insa aplicatia nu v-a functia , daca de expmlu la unul din acei timpi vei iintroduce ‘a’, v-a aparea mesajul ca toate campurile trebuie copletate, deoarece compilatorul in momentul in care intalneste altceva in afara de int v-a intra pe ramura catch, iar pe aceasta ramura eu am am pus sa imi afiseze acest mesaj.

# 5. Testare

Mai sus am explicat pas cu pas ce mesaje de eroare o sa apara in momentul in care clientul foloseste gresit aceasta aplicatie. Daca utilizatorul v-a utiliza correct aplicatia nu v-a fi nicio problema. Trebuie luat in clacul ca clientii mei sunt generate random, deci s-a r putea ca in anumite momente sa imi apara client cu acelasi numar, deoarece eu generez clientii in intervalul 1-50.

# 6. Rezultate

La sectiunea 2.3 am introdus screenshot-uri in care avem efectuata fiecare operatie. De remarcat cateva observatii:

Dupa cum am specificat mai sus pot aparea client de doua ori, insa aceasta nu reprezinta o problema. Cleintii mei sunt generate random, deci mie nu o sa imi apara clientul 1, 2, 3 si asa mai departe, imi v-a aparea clientul 34 dupa care clientul 48 de exeplu sau clientul 10. Eu am ales aceasta metoda deoarece am luat in considerare faptul ca cleintii pot sa petreaca mai mult timp in magazin fata de un client care a ajuns dupa el. De exemplu vine clientul 4 la secunda 3, dupa care clientul 5 la secunda 4, clientul 5 poate sa fie la coada inainte clientul 4.

# 7. Cocluzii

Dupa realizarea acestui proiect am aprofundat termenii de thread si multithreading folosindu-I in aceasta aplicatie. Cu toate ca stiam ce inseamna acesti termini am reusit sa ii folosesc in proiectul meu, nu pot spune ca a fost usore, insa acum am inteles cum se folosesc. Pe langa acest lucru am invatat si despre LinkedList si am aflat destul de multe metode prin care poti lucre foarte usor cu aceste liste.

# 8.Bibliografie

<https://cloud.smartdraw.com/editor.aspx?templateId=95dfa214-2a45-46dd-857e-b2dc7286f844#depoId=9120997&credID=-18736549>

Acest site te ajute intr –un mod foarte simplu sa realizezi diagram UML.

<http://www.cs.ubbcluj.ro/~forest/HtmlFolder/PDPJ/threads.html>

<http://inf.ucv.ro/documents/tudori/laborator8_53.pdf>

<https://ro.wikipedia.org/wiki/Model-view-controller>