

Monitorizarea activitatii unui magazin**1 Introducere - prezentarea problemei**

În cadrul temei date, am considerat un magazin care vinde fructe către clienți. Problema implementată reprezintă desfășurarea activității acestuia în condiții normale de zi cu zi. Prin aceasta, dorim să realizăm achiziționarea de produse de către clienți, aprovizionarea magazinului cu produse noi și totodată monitorizarea continuă a produselor de pe stoc.

Ca limbaj de programare pentru această documentație și exemplu specific am utilizat Java. Acest lucru nu a pus o problemă din punct de vedere al hardware-ului laptopului, acesta având 8GB RAM, procesor Intel Core i7, a 9-a generație.

1.1 Pas 1. Definire problemă

Să se implementeze o aplicație formată din 3 taskuri:

- task de tip client (achiziționează produse)
- task de tip server (monitorizează stocul din magazin)
- task de tip aprovizionare (aprovizionează cu produse)

Pentru o bună desfășurare a activității magazinului avem următoarele condiții:

- doi clienți nu pot comanda simultan;
- numărul de mere achiziționate trebuie să fie mai mic sau egal decât stocul disponibil;
- în cazul în care nu mai există produse, sau cererea este mai mare decât piața se face aprovizionarea în funcție de dorința utilizatorului

2 Analiza problemei

Se dorește achiziționarea continuă de fructe din magazin până la punctul în care nu mai există niciun produs pe stoc. Pentru aceasta clienții trebuie să poată comanda unul, câte unul, fără a se suprapune niciodată. Are deci voie să comande un singur client odată. Pentru a se putea face tranzacțiile trebuie să ne asigurăm constant că există un număr de produse pe stoc mai mare decât cele solicitate, monitorizare făcută de taskul "Tranzacții". Utilizatorul are de asemenea posibilitatea de a aproviziona cu mere magazinul, acest lucru fiind posibil doar atunci când numărul de mere dorit de client este mai mare decât soldul disponibil.

2.1 Pas 2. Analiza cerințelor

Secvențe posibile:

- La fiecare iteratie clientul va achizitiona un anumit numar de mere pana cand soldul devine 0 sau pana cand nu mai sunt suficiente produse pe stoc
- Aprovizionarea se realizeaza cand soldul este mai mic decat cerinta
- Monitorizarea soldului si a cerintei se fac pe tot parcursul programului prin taskul "Tranzactii"

Secvențe imposibile:

- Doi clienti nu pot sa achizitioneze produse simultan - nu indeplineste conditia 1
- Aprovizionarea nu se poate realiza atunci cand stocul este mai mare sau egal cu cerinta clientului - nu indeplineste conditia 3
- Nu se pot achizitiona produse atunci cand cerinta e prea ridicata - nu indeplineste conditia 2

3 Definirea structurii aplicației

Structura aplicatiei este impartita in 3 etape importante. Acestea asigura buna functionare si desfasurare a actiunilor in cadrul magazinului Etapele sunt:

- Se alege un numar de mere pentru a fi achizitionat de catre client din magazin. Aceasta tranzactie este repetata continuu pana cand cerere excede numarul de produse de pe stoc. Avem grija totodata ca doi clienti sa nu isi suprapuna comenzile. Aceasta parte reprezinta excluderea mutuala, clientii putand sa acceseze metode diferite simultan, dar ne fiind posibil pentru acestia sa acceseze aceeasi metoda in acelasi timp.
- Taskul "Tranzactii" monitorizeaza permanent desfasurarea tranzactiilor avand grija ca numarul de mere de pe stoc sa nu devina negativ in urma vreunei tranzactii. Totodata aici ne asiguram ca odata ce cererea este mai mare decat numarul de mere se va realiza aprovizionarea.
- Aprovizionarea se face in momentul in care nu mai sunt suficiente produse pe stoc, urmand ca utilizatorul sa introduca suma pe care vrea sa o introduca in depozit.

3.1 Pas 3. Definirea taskurilor care compun aplicația

În cazul exemplului prezentat, acestea sunt taskurile:

- task de tip client (achizitioneaza produse): Client
- task de tip server (monitorizeaza stocul din magazin):Server
- task de tip aprovizionare (aprovizioneaza cu produse): Aprovizionare

4 Definirea soluției în vederea implementării

Pentru a realiza aplicatia creata am utilizat mediul de programare Java, pentru editor utilizand Eclipse.

Am inceput prin crearea a trei clase principale (super class) pe care le-am denumit "ProgramPrincipal", "Tranzactii" si "Cont" in cadrul carora am utilizat diferite metode si functii pentru a asigura buna desfasurare a taskurilor.

Vom aborda fiecare clasa separat.

Program principal - Aici se afla main-ul aplicatiei unde am creat doua threaduri, reprezentand clientii prin intermediul metodei continute deja in Java, "Thread". Tot in cadrul acestei clase am realizat un constructor de tip Cont din cadrul clasei "Tranzactii" pe care il utilizam in crearea obiectului "mentenanta" utilizat pentru cele 2 threaduri.

Cont - Aici se afla toate actiunile pe care le desfasoara cineva in raport cu produsele din magazin. Avem numarul de produse de pe stoc declarate in program. Bineinteles, acestea ar fi putut fi date de la tastatura, dar pentru simplitate am declarat in cadrul acestei clase. De asemenea, tot aici avem si functiile pentru aprovizionare si achizitionare de mere unde doar se aduna sau se scade dupa caz la cantitatea totala de produse din magazin.

Tranzactii - Acesta este practic serverul nostru care monitorizeaza constant activitatea in aplicatie. Avem aici un task de tip runnable care se executa continuu pana la indeplinirea unei conditii. In cazul nostru, aceasta conditie este data de variabila "exit" care este setata atunci cand cerinta este mai mare decat numarul de produse din magazin. In cadrul functiei run avem doua functii while care asigura desfasurarea continua a aplicatiei, totodata aici putand a fi citite de la tastatura cantitatea de mere pe care utilizatorul doreste sa o retraga din magazin. Toate aceste introduceri de la tastatura sunt anuntate pe ecran prin text. Pentru citire am utilizat "scanner", dand import in Java la biblioteca corespunzatoare. De aici este apelata functia cumparaMere, avand ca parametru cantitatea de mere introdusa de la tastatura de la utilizator. Functia cumparaMere este utilizata pentru realizarea excluderii mutuale, doi clienti ne putand sa realizeze simultan o tranzactie. Totodata aici avem si conditia de timp, data de:

```
try Thread.sleep(3000); catch (InterruptedException ex)
```

In continuarea programului trecem la etapa in care nu mai sunt suficiente produse pe stoc si utilizatorul este interogat daca doreste sa aprovizioneze magazinul sau nu. (DA -1/Nu -0). Daca raspunsul este 0, programul se incheie afisand un mesaj pe ecran, in timp ce daca raspunsul este 1, se solicita numarul de mere care se doreste a fi introdus si programul ruleaza de l capat.

4.1 Pas 4. Soluție de implementare

Aleg mecanismele de sincronizare și comunicare între taskuri, prezentând organigramele taskurilor (e.g. în Fig. 4).

- aleg 2 taskuri de achizitionare si aprovizionare, cu val. inițiale Client = 0, Achiz = 1
- aleg 1 task de tip monitorizare, cu val. inițială 0, Sum = 0

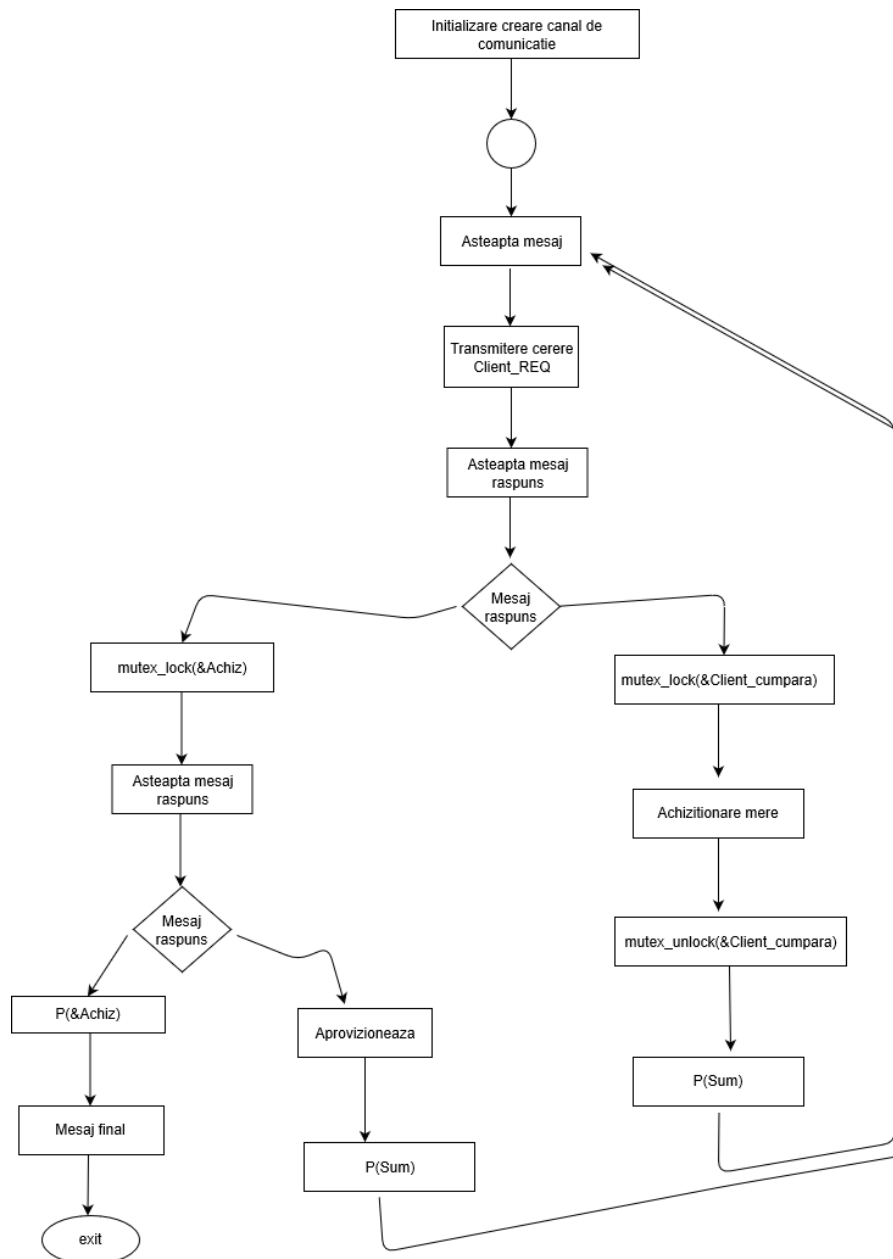


Figure 1: Soluție implementare - organigrame taskuri

5 Implementarea soluției

În cele ce urmează, voi atașa codul aplicației, aceasta având comentariile necesare. Modul în care funcționează programul a fost explicat mai sus, la punctul ”Definirea soluției în vederea implementării”.

5.1 Cod program

Listing 1:

```
//PROGRAM PRINCIPAL
```

```

package Pachet3;

import java.util.Scanner;

import Pachet1.Cont;
import Pachet2.Tranzactii;

public class ProgramPrincipal {

    public static void main(String[] args) {

        //Realizarea threadurilor
        Cont cont = new Cont(); //Constructor
        Tranzactii mentenanta = new Tranzactii(cont);
        Thread t1 = new Thread(mentenanta, "Cumparator0");
        Thread t2 = new Thread(mentenanta, "Cumparator1");

        t1.start();
        t2.start();
    }
}

//PROGRAMUL DE MONITORIZARE - excludere mutuala, sincronizare si comunicare intre taskuri

package Pachet2;
import Pachet1.Cont;
import Pachet2.creamThread;

import java.util.Scanner;
import java.util.concurrent.TimeUnit;

public class Tranzactii implements Runnable {
    private Cont cont;
    private volatile boolean exit = false;
    int depunere_boolean=1;

    /*public class creamThread{
    creamThread myClass = new creamThread();
    String numeOm = this.passengerName;
    }*/

    public Tranzactii(Cont account) {
        this.cont = account;
    }

    int numarMereCumparate;
    public void run() {
        while(!exit) { //variabila care determina daca programul continua rularea sau nu

```

```

while(!exit) {
    int numar_mere_1;
    //int numar_mere_2;
    System.out.println("Ziua buna! Cate mere ati dori sa cumparati de la noi?");
    Scanner scanner = new Scanner(System.in);
    numar_mere_1 = scanner.nextInt();
    cumparaMere(numar_mere_1);
    //Scanner scanner_2 = new Scanner(System.in);
    //numar_mere_2 = scanner_2.nextInt();
    cumparaMere(numar_mere_1);
    if (cont.getBalance() < 0) {
        System.out.println("Momentan am ramas fara mere pe taraba!");
    }
}

private synchronized void cumparaMere(int numarMereCumparate) {
    int i=0;
    int depunere_mere;
    while(!exit) {
        //Verificarea balance-ului produselor, daca indeplineste sau nu conditia
        if (cont.getBalance() >= numarMereCumparate /*|| cont.getBalance() >= numarMereCumparate_2*/) {
            System.out.println(Thread.currentThread().getName()+i + ": As vrea si eu "+numarMereCumparate + " me
            try {
                Thread.sleep(3000);
            } catch (InterruptedException ex) {
            }
            cont.taieDinMere(numarMereCumparate);
            System.out.println("Vanzator: Poftiti "+numarMereCumparate+ " mere. O zi buna!");
            if(i==1) {
                i--;
            } else i++;
            } else {
                System.out.println("Ne pare rau, " + Thread.currentThread().getName()+i + ", nu avem suficiente mer
                + cont.getBalance()+" mere...");

                //In cazul in care nu se indeplineste conditia aprovizionam cu produse noi

                // Adaugam mere
                System.out.println("A ajuns aprovizionatorul. Vreti sa ne aprovizionati cu mere? DA = 1/NU = 0");
                Scanner scanner = new Scanner(System.in);
                depunere_boolean = scanner.nextInt();

                if(depunere_boolean == 1) { //variabila care determina daca userul mai vrea sa depoziteze mereu sau
                    //Cu cate mere aprovizionam magazinul
                    System.out.println("Multumim! Cate mere vreti sa ne donati?");
                    Scanner scanner1 = new Scanner(System.in);
                    depunere_mere = scanner1.nextInt(); //numarul de mere depozitate

                    cont.depozitare(depunere_mere);
                    System.out.println("Mai avem " + cont.getBalance()+ " mere!");
                    run();

```

```

}
else if( depunere_boolean == 0) {
System.out.println("Multumesc ca ati cumparat de la noi! ");
exit = true; //variabila care determina daca programul continua rulara sau nu
}}
}
}
}

//CONT unde se fac functiile de achizitionare, gestiune produse si aprovizionare
package Pachet1;

public class Cont {
private int numarMerePeTaraba = 60;

//Cate mere sunt
public int getBalance() {
return  numarMerePeTaraba;
}

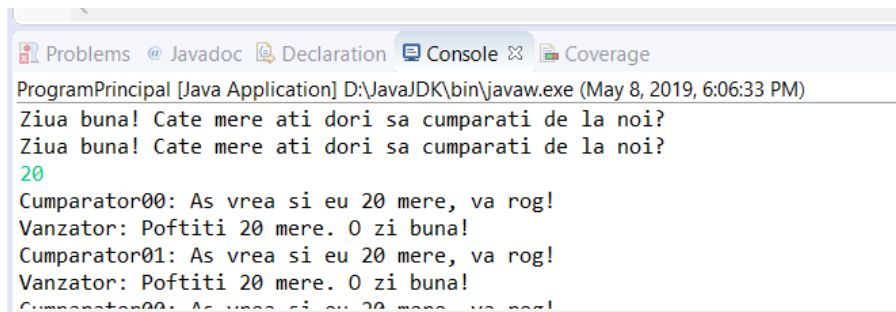
//Aprovizioneaza magazinul
public void depozitare(int cantitate) {
numarMerePeTaraba=numarMerePeTaraba + cantitate;
}

//Retragere mere
public void taieDinMere(int cantitate) {
if(cantitate<= numarMerePeTaraba) {
numarMerePeTaraba=numarMerePeTaraba - cantitate;
}
else {
System.err.println("Nu mai avem mere pe stoc.");
}
}
}
}

```

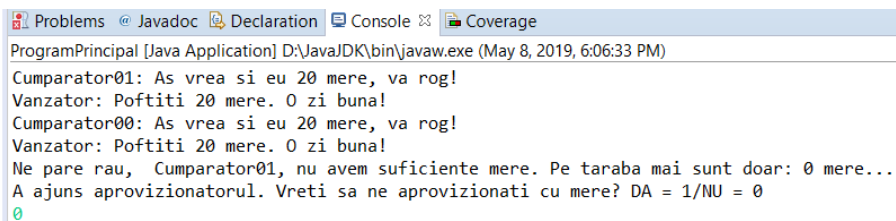
6 Testarea aplicației si validarea soluției propuse

Pentru aplicatie este totul foarte intuitiva odata ce dam run. Desi apare de 2 ori prima cerinta, trebuie introdusa o singura valoare. Pentru o rulare optima trebuie urmate intocmai indicatiile specificate pe ecran la data rularii. Acestea sunt imaginile cu cazurile posibile ale aplicatiei:



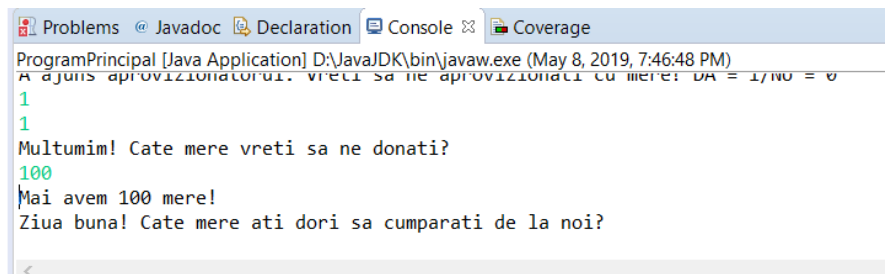
```
Problems @ Javadoc Declaration Console Coverage
ProgramPrincipal [Java Application] D:\Java\jdk\bin\javaw.exe (May 8, 2019, 6:06:33 PM)
Ziua buna! Cate mere ati dori sa cumparati de la noi?
Ziua buna! Cate mere ati dori sa cumparati de la noi?
20
Cumparator00: As vrea si eu 20 mere, va rog!
Vanzator: Poftiti 20 mere. O zi buna!
Cumparator01: As vrea si eu 20 mere, va rog!
Vanzator: Poftiti 20 mere. O zi buna!
Cumparator02: As vrea si eu 20 mere, va rog!
```

Figure 2: Desfasurarea programului.



```
Problems @ Javadoc Declaration Console Coverage
ProgramPrincipal [Java Application] D:\Java\jdk\bin\javaw.exe (May 8, 2019, 6:06:33 PM)
Cumparator01: As vrea si eu 20 mere, va rog!
Vanzator: Poftiti 20 mere. O zi buna!
Cumparator00: As vrea si eu 20 mere, va rog!
Vanzator: Poftiti 20 mere. O zi buna!
Ne pare rau, Cumparator01, nu avem suficiente mere. Pe taraba mai sunt doar: 0 mere...
A ajuns aprovizionatorul. Vreti sa ne aprovizionati cu mere? DA = 1/NU = 0
0
```

Figure 3: Refuzarea aprovizionarii magazinului.



```
Problems @ Javadoc Declaration Console Coverage
ProgramPrincipal [Java Application] D:\Java\jdk\bin\javaw.exe (May 8, 2019, 7:46:48 PM)
A ajuns aprovizionatorul. Vreti sa ne aprovizionati cu mere? DA = 1/NU = 0
1
1
Multumim! Cate mere vreti sa ne donati?
100
Mai avem 100 mere!
Ziua buna! Cate mere ati dori sa cumparati de la noi?
```

Figure 4: Aprovizionarea magazinului.