Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Simulator de cozi

Documentatie – tema 2 –

Czako Zoltan

Grupa: 30225

Contents

1. Obiectivul temei :

Enuntul temei :

„ Proiectati si implementati un program simulator pentru procesare cozilor ”

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

Cozile apar atat in viata reala cat si in modele si concepte din domeniul calculatoarelor. Scopul principal al cozilor este de a asigura un loc pentru un client inainte ca acesta sa fie servit. Managementul cozilor se ocupa de minimizare timpului de asteptare al clientilor in coada. O metoda de minimizare a timpului de asteptare consta in introducerea mai multur servere, adica mai multe cozi in sistem (fiecare coada este considerat ca are un singur procesor), dar in cazul acesta creste si costurile. Daca o coada noua este adaugata, atunci clietii sunt partajate in mod egal intre cozi.

Programul trebuie sa simuleze o serie de clienti care trebuie astept pentru a fi servit, clientul intra in coada, asteapta, este servit si la final paraseste coada. Pentru a calcula timpel mediu de asteptare, trebuie sa stim timpul de sosire, timpul de plecare plecare si timp necesare pentru servire. Timpul de sosire si timpul de servire depinde de fiecare client, separat – clientul alege cat vrea sa cumpere si cand vrea sa intra in coada petru a plati produsele alese. Timpul final depinde de numarul cozilor, numarul clientilor si serviciile dorite de clienti.

Date de intrare:

- Timpul minim si timpul maxim de sosire intre clienti
- Numarul minim de servicii si numarul maxim de servicii
- Numarul cozilor
- Timpul simularii
- Alte informatii (am ales sa putem alege partea zilei pentru care facem simularea, dimineate, dupa-masa, seara si noaptea)

Date de iesire:

- Timp de asteptare mediu, considerand timpul de servire pentru fiecare client si numarul cozilor
- Log of events and main system data
- Evolutia cozilor
- Ce se intampla daca – analizarea simularii

3. Proiectare (diagrama UML, structuri de date, proiectare clase , interfete , relații, packages, algoritmi, interfața utilizator)

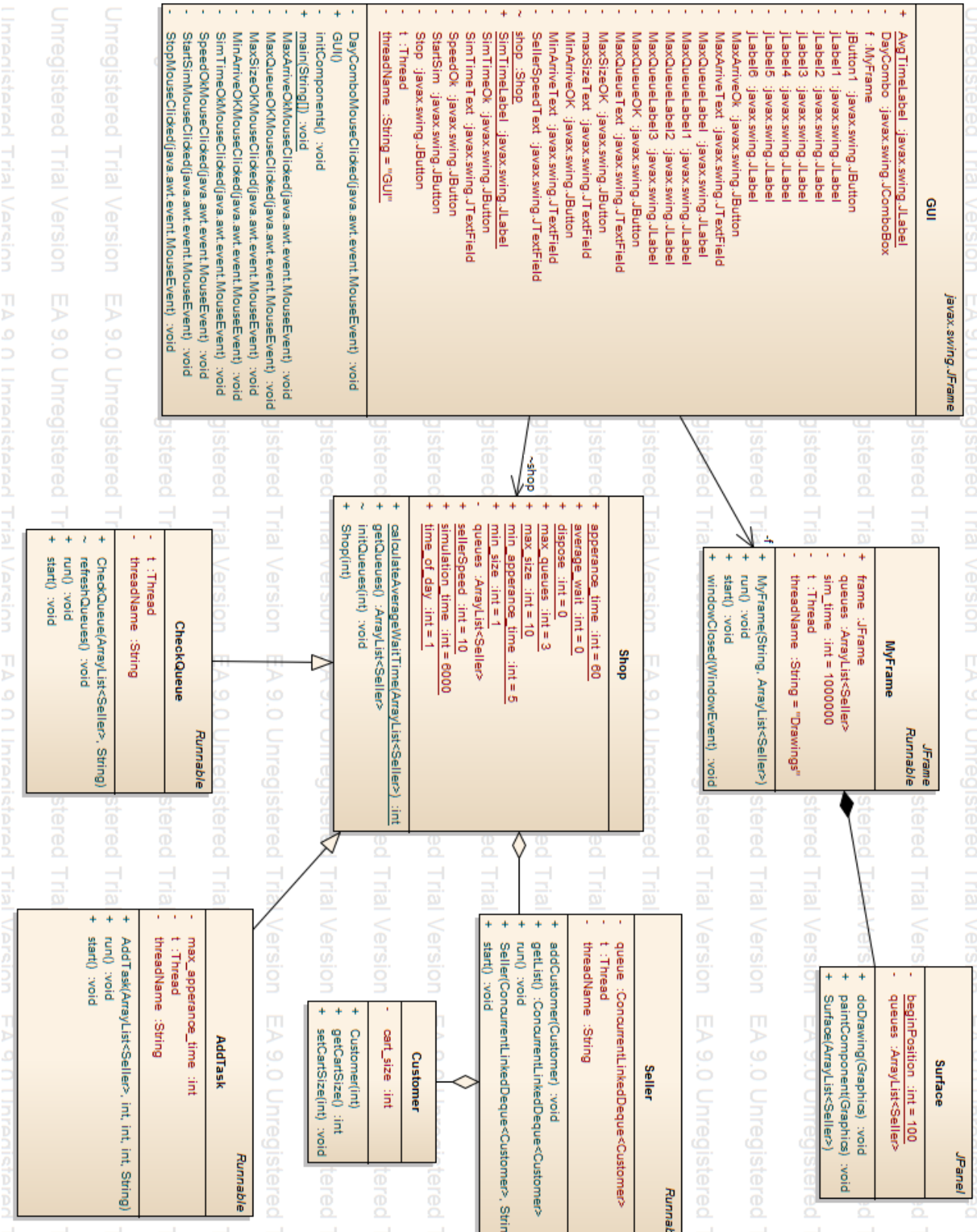
Pentru realizarea acestui proiect dintre conceptele de OOP am folosit mai ales compozitia si agregarea, dar si mostenirea. Mostenirea am folosit pentru partajarea memoriei intre mai multe clase. O tehnica importanta de realizare al acestui proiect consta in folosirea Thread-urilor.

Pentru fiecare coada am folosit un thread, care verifica daca mai este cineva in coada si daca exista un client, atunci incepe sa il serveasca, adica incepe sa scoata obiectele din cosul clientului. Daca clientul nu mai are nimic in cos si a platit pentru toate produsele, atunci scoate clientul din coada si incepe servirea clientului urmator. Am folosit un alt thread pentru „creearea" clientilor, si partajarea lor in mod egal printre cozi. Acest thread creeaza un client cu marimea cosului in intervalul cu capetele timp minim de servire si timp maxim de servire si alege intotdeauna coada cu numarul minim de clienti. Un alt thread am folosit pentru reorganizarea dinamica a cozilor, adica in cazul in care una sau mai multe cozi se inchid sau se deschid, clientii sunt partajate in mod egal, pentru ca timpul mediu de asteptare sa fie cat mai mic. Inca un thread folosesc pentru afiseara dinamica pe ecran a evolutiei cozilor, unde un cerculet reprezinta un client, numarul din cerc reprezinta timpul necesar de servire iar patratul reprezinta procesorul sau casa.

Clasele folosite:

- Shop – clasa care contine cozile, totul se intampla in magazin, adica magazinul este supercalas
- Customer - clasa folosita pentru creearea unui client, cuprinde marimea cosului
- Seller - clasa care se ocupa de servirea clientilor, cuprinde o coada de clienti (o coada sincronizata pentru folosirea corecta a thread-urilor)
- AddTask - clasa folosita pentru adaugarea clientilor si partajarea lor in proportie egala
- CheckQueue – clasa folosita pentru verificarea lungimii cozilor, iar in cazul in care cozile nu sunt proportionale, reorganizeaza cozile, trimite clientii din coada cea mai lunga in coada cu numarul minim de clienti
- Surface – clasa care deseneaza pe un suport grafic interiorul magazinului, adica cozile si clientii din fiecare coada
- MyFrame – clasa care creeaza un frame suport pentru grafica facuta de Surface
- GUI – e interfata grafiaca pentru panoul de control, prin care putem controla datele de intrare, putem schimba cozile si putem manipula evoluta cozilor

Diagrama UML :



Descrierea claselor in detaliu :

Clasa Shop are doua metode importane , si anume calculateAverageWaitTime , metoda ce calculeaza timpul mediu de asteptare , prin calcularea sumei pentru fiecare client din fiecare coada si imparte la numarul cozilor existente.

A doua metoda importanta e metoda numita setLists , care este apelata de fiecare data, cand facem o modificare a numarului de cozi. Aceasta metoda , impreuna cu metoda din clasa CheckQueue , si anume refreshQueues realizeaza partajarea corecta a clientilor , chiar daca se schimba numarul cozilor , sau in caz de eroare , daca o coada se blocheaza.

```
void refreshQueues( )
  {
      int min = 100000 , max = 0 ;
      int ind_min = 0 , ind_max = 0 ;
      int i = 0 ;
      for( Seller queue : this.queues )
      {
         if( queue.getList( ).size( ) < min )
            {
               min = queue.getList( ).size( ) ;
               ind_min = i ;
            }
         if( queue.getList( ).size( ) >  max )
            {
               max = queue.getList( ).size( ) ;
               ind_max = i ;
            }
         i++ ;
      }

      while( Math.abs( max-min ) > 1 && ( min <  max) )
      {
         Customer c;
         c = queues.get( ind_max ).getList( ).pollLast( ) ;
         queues.get( ind_min ).getList( ).add( c ) ;
         max-- ;
      }
  }
```

```
public static ArrayList<Seller> setLists( ArrayList<Seller> list ,  int max_list )
   {
      int size,i;
      ConcurrentLinkedDeque<Customer> last  ;
      size = list.size( ) ;
      Customer c;
      if ( size > max_list ){
          while( size > 0 && size > max_list )
          {
             last = list.get(size-1).getList( ) ;
             i=0;
             for( Customer cus : last)
             {
                if( i == size – 1 ) i =  0;
                //c  = last.pop();
                if ( cus.getCartSize( ) >= 1 )
                   list.get( i ).addCustomer( cus );
                i++ ;
             }

             if ( size > 0 ){
                size-- ;
                list.remove( size ) ;
             }
          }
           // return list ;
       }
          else
          {
             while( size < max_list )
             {
                ConcurrentLinkedDeque<Customer> deq = new
ConcurrentLinkedDeque<Customer>( ) ;
                Seller s = new Seller( deq , "new" );
                list.add( s ) ;
                size++ ;
                i=0 ;
                for(Seller sel : list)
                {
                 if (!sel.getList().isEmpty() && i!=size-2)
                 {
                  Customer cust = sel.getList( ).getLast( ) ;
                  if(cust.getCartSize( ) >= 1 )
                      list.get( size - 1 ).addCustomer( cust ) ;
                 }
                  i++ ;
```

```
                    }
                    s.start ( ) ;
                }
                // return list ;
            }
        return list;
    }
```

Clasa Seller are o lista de Clienti (adica un ConcurrentLinkedDeque<Customer>) si realizeaza serviciile pentru clienti, scoate din cosul lor obiectele secvential si la final clientul poate sa plece.

O alta clasa importanta este clasa Surface , care realizeaza reprezentarea grafica a cozilor, reprezinta evolutia in timp real a cozilor , astfel reprezentad grafic un magazin intreg.

Metoda cea mai importanta care prin cateva calcule matematice deseneaza clientii si casele. Aceasta metoda este implementata astfel :

```
public void doDrawing ( Graphics g ) {

    Graphics2D g2d = ( Graphics2D ) g ;

    int size = this.queues.size( ) ;


    for(int i=0; i < size; i++)
    {
       g2d.drawRect( i * 40 + this.beginPosition - 40 , 20 , 30 , 30 ) ;
       for (int j=0 ; j < this.queues.get( i ).getList( ).size(); j++ )
       {
          g2d.drawOval(this.beginPosition + i * 40 – 40 ,  ( j + 2 ) * 30 , 30 , 30 ) ;
       }
       int j=0 ;
       ConcurrentLinkedDeque<Customer> dq = this.queues.get( i ).getList( ) ;
       Iterator<Customer> itr = dq.iterator( ) ;
       while ( itr.hasNext ( ) ) {
          Customer c = itr.next( ) ;
          Integer nr = c.getCartSize( ) ;
          String text;
          text  =  nr.toString( ) ;
          g2d.drawString( text , this.beginPosition + i * 40 + 10 – 40 , ( j + 2 )  * 30 + 20 );
          j++  ;
       }
    }
 }
```

Unde cercul reprezinta un client, iar patratul reprezinta casa.

MyFrame – clasa care creeaza un frame suport pentru grafica facuta de Surface

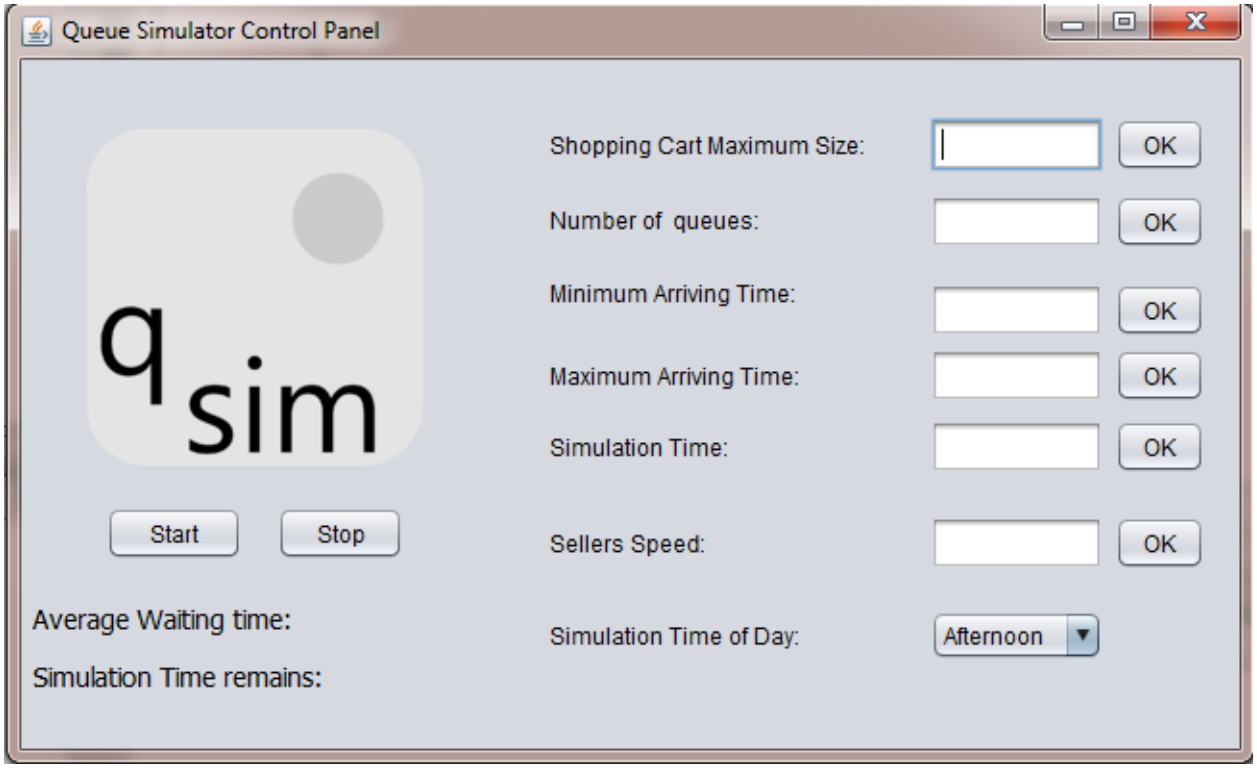
GUI – e interfata grafiaca pentru panoul de control, prin care putem controla datele de intrare, putem schimba cozile si putem manipula evoluta cozilor. Cuprinde Event Handler-urile pentru fiecare buton, adica pentru butoanele Ok si butonul Start si Stop, dar is declararea textfieldurilor si se ocupa de adunarea datelor referitoare la simulare, adica ofera o interfata prietenoasa si usor de folosit pentru un utilizator oarecare.

Interfata grafica este alcatuita din doua componente mari, adica Control Panel-ul pentru introducerea datelor, adica pentru introducerea timpului minim si timpului maxim de asteptare, numarul serviciilor maxim si numarul serviciilor minim, timpul simularii, partea zilei pentru care vrem sa facem simularea si numarul cozilor. Tot pe control panel apare si timpul mediu de asteptare si timpul ramas din simulare.

Componenta a doua reprezinta reprezinta grafic interiorul magazinului, aici putem vizualiza in timp real evolutia cozilor, miscarea clientilor, aparitia clientilor si iesirea clientilor din coada, sau inchiderea sau deschiderea unei sau a mai multor cozi, si repartizarea clientilor.

Interfata grafica:

Control Panel-ul:

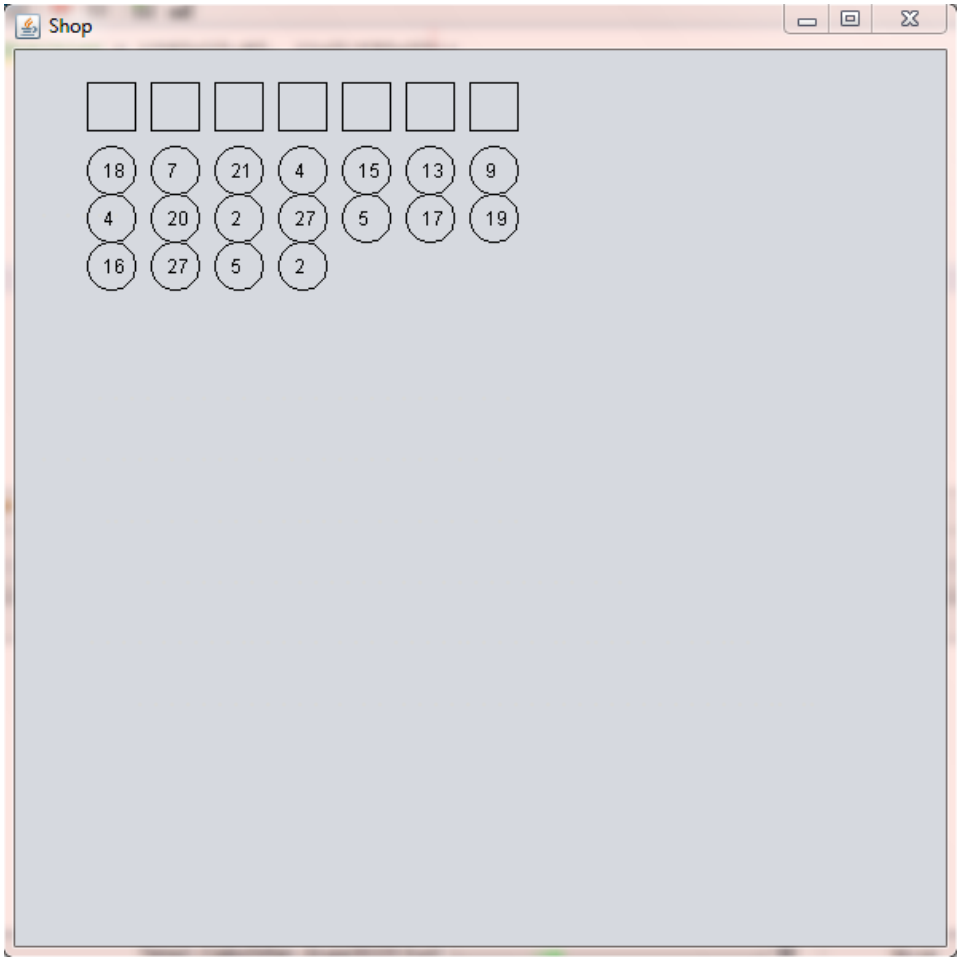


Datele introduse trebuie sa fie confirmate. Confirmarea se face apasand butonul OK. Daca informatia introdusa e corecta, atunci textfieldul se goleste, altfel se va lua in considerare valoarea default.

Pentru a alege partea zilei pentru care facem simularea, am introdus un combobox cu valorile Morning - - - dimineata , Afternoon - - - dupa - masa , Evening - - - seara si Night - - - noaptea.

Daca vrem sa incepem simularea, trebuie sa dam click stanga pe butonul Start, in urma careia se va deschide o noua fereastra, in care se vor misca clientii in functie de datele introduse. Daca apasam butonul Stop , atunci simularea se opreste si putem sa incepem o simulare noua.

Dupa apasaea butonului start , va aparea urmatoarea fereastra:



Aici cercul reprezinta clientul, numarul din cerc reprezinta obiectele ramase in cos, iar patratul reprezinta casa(procesorul).

4. Testare

Aceasta aplicatie a fost testata de mai multe ori. Testarea am facut prin introducerea manuala a datelor, verificand si cazurile particulare (de exemplu daca avem o singura coada, daca avem un timp de simulare foarte mica sau foarte mare, daca avem timp de serviciu foarte mare sau foarte mica , daca e noapte sau e dimineata , daca scoatem o coada sau daca mai ntroducem o coada, daca introducem mai multe cozi deodata sau scoate mai multe cozi deodata etc.) iar rezultatul testarii a fost intotdeauna conform realitatii, fara greseli.

Pentru o testare mai buna am facut aple si la alte persoane, care au incercat utilizarea aplicatiei cautand greseli, iar daca au aparut ceva greseli pe care pana atunci nu am luat in considerare, pe care pana atunci nu am gasit, am reformulat codul pentru corectarea acestor greseli. In prezent aplicatia merge fara greseli.

5. Rezultate

După testarea programului vedem că programul functionează perfect, verifică datele de intrare , afișează rezultatele pe interfața grafică, si efetuează comenzile primite de la utilizatorul programului.

6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

Lucrand la acest proiect am invatat folosirea cozilor, si a threadurilor, am invatat multithreadingul si conceptele ce se leaga de acesta. Am invatat doua metode de folosire a threadurilor, si anume prin extinderea clasei Thread, dar aceasta metoda nu mi s-a parut destul de eficient, deaorece o clasa poate sa mosteneasca doar de la o singura superclasa , iar in cazul in care extindem clasa Thread, atunci nu mai avem posibilitatea sa extindem si alte clase , deci aceasta metoda este o metoda foarte limitata. In aceasta cauza am evitat folosirea acestei metode si am gasit o metoda mai eficienta, adica implementez interfata Runnable, care are o singura metoda , si anume metoda run() . Daca suprascriem aceasta metoda dupa preferintele noastre , atunci threadul nostru va face exact ce am scris in run() . Un mare avantaj al folosirii interfetei runnable consta in faptul ca , putem implementa mai multe interfete si ne mai ramane si posibilitatea de mostenire, adica clasa noastra va fi mult mai felxibila.

Pe langa threaduri, am invatat cum sa folosesc un obiect de tip Graphics sau Graphics2D, si am invatat desenarea diferitlor forme geometrice, si calcularea pozitiei acestora in functie de coordonatele x si y.

Pentru realizarea acestui proiect ar trebuit sa citesc foarte multe articole stiintifice, posturi de programatori (mai ales pe StackOverflow). Consider ca, lucrand pe acest proiect am inteles mult mai bine conceptele de OOP, folosirea threadurilor, desenele grafice si interfata swing. Pe de alta parte am inteles conceptul de multitasking si concurenta programelor , care pana acum era doar o teorie pentru mine , dar acum am folosit si practic si am inteles cat de important si cat de folositor este aceasta abordare a programelor , a proceselor si a taskurilor.

7. Bibiliografie

Carte : Thinking in java – fourth edition Bruce Eckel President, MindView, Inc.

Internet :

https:// www. youtube . com / - turoriale

http:// stackoverflow . com /

http:// docs. oracle. com / javase / tutorial /

http:// www. java2s . com / Code / JavaAPI / javax . swing/JPanelpaintComponentGraphicsg.htm

http:// docs.oracle.com / javase / tutorial / essential / concurrency /

http:// www. Tutorialspoint . com/ java / java_multithreading.htm