CALCULATOR DE POLINOAME

Buciuman Mihai Catalin
Grupa 30227

Cuprins

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Testare
6. Rezultate
7. Concluzii si Dezvoltari Ulterioare
8. Bibliografie

1. Obiectivul Temei

Obiectivul principal al acestei teme este de a proiecta un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi .

Se doreste citirea de la tastatura a doua polinoame sub forma 2X^3+2X^2+X+1, astfel incat sa se poata realiza asupra lor operatii de adunare, scadere, inmultire ,impartire cat si operati de derivare si integrare. Calculatorul de polinoame dispune de o interfata grafica implementata cu ajutorul framework-ului JavaFX acesta aducand un plus aspectului. Interfata este implementata cat mai simplu fiin “User Friendly”.

Obiectivele secundare ale acestei teme sunt de a implementa un grafic prin care sa se poata exprima grafic fiecare polinom.

2. Analiza Problemei

Un polinom este o expresie construita dintr-una sau mai multe variabile și constante, folosind doar operatii de adunare, scadere, inmulțire și impartire.

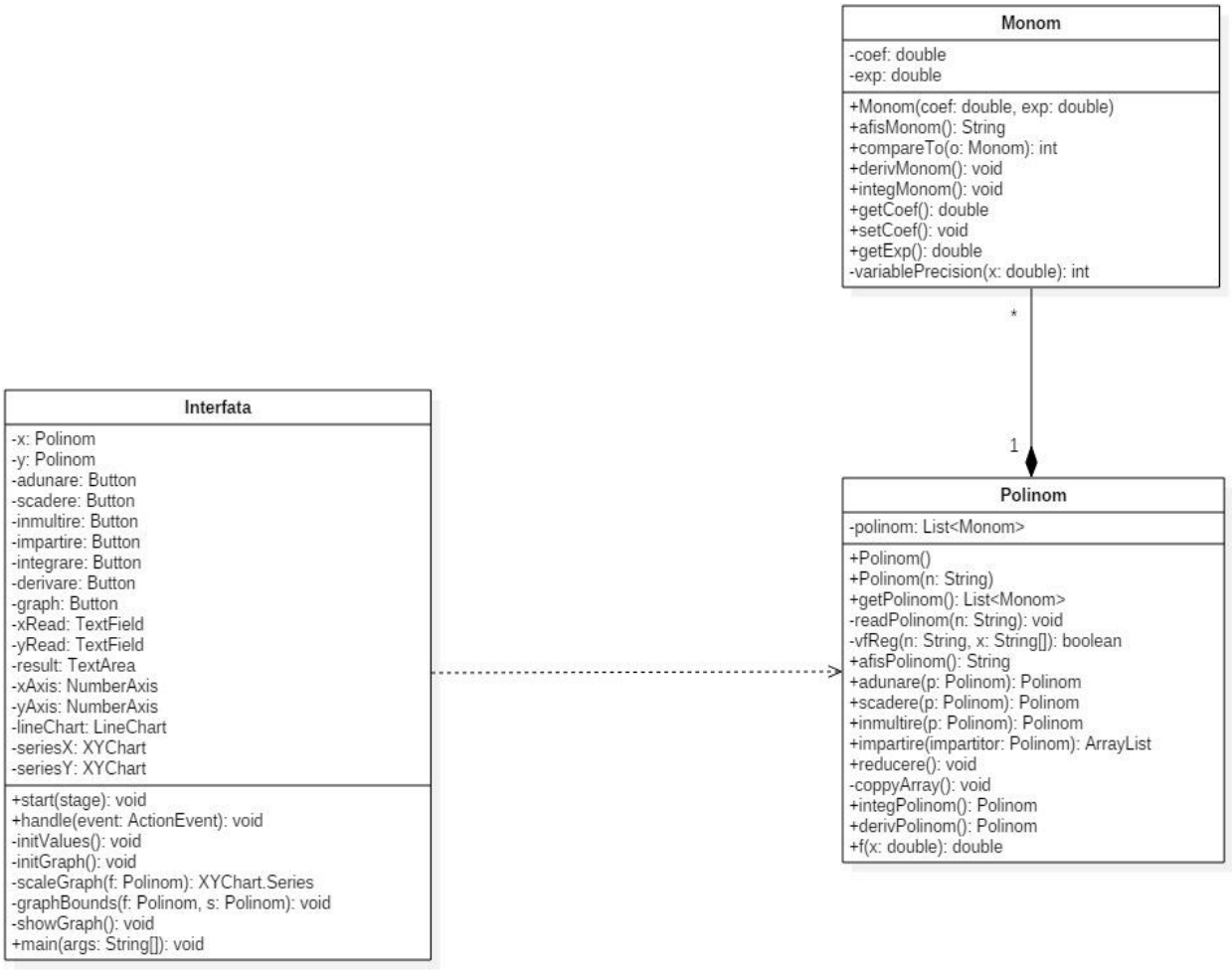
Ex: X^ 2 – 4X + 7 este un polinom

Polinomul este format din unul sau mai multe monoame .

Ex: 2X^2 este monom

Modul de abordare al proiectari acestui calculator este de a imparti polinomul introdus de utilizator in mai multe monoame astfel incat sa fie procesat mai efficient.

3. Proiectare



Pentru a proiecta cat mai eficient sa ales o metoda de a imparti in mai multe monoame polinomul introdus de utilizator . Calculatorul se foloseste de doua clase pentru procesarea polinoamelor . Clasa Polinom este cea care va prelua toate monoamele si in care se vor procesa intstructiunile de adunare , scadere , inmultire ,impartire ,derivare si integrare . Pentru reprezentarea polinoamelor in reper cartezian s-a folosit metoda f() pentru a calcula f(x) ca la o functie oarecare.

Clasa Polinom retine cu ajutorul unui List<Monom> toate monoamele.

Monoamele sunt formate dintr-un coeficient de tipul double si de un exponent de tip double

Am ales tipul double pentru a se putea scrie mult mai usor rezultatul integrari sau al impartiri in cazul in care un coeficient ajunge sa fie sub forma de fractie (pt ½ se va afisa 0.50)

Clasa Interfata se ocupa de partea grafica mai exact de GUI-ul acestui calculator de polinoame . GUI-ul este compus din doua TextField-uri in care utilizatorul poate introduce cele doua polinoame pentru a putea fi procesate . Interfata contine si butoanele pentru operati cat si un buton pentru generarea graficului .

Atentie ! Graficul generat este doar sa dea utilizatorului o idee despre cum ar arata functia , nu este facut astfel incat sa specifice exact fiecare curba sau punct de intersectie

4. Implementare

Proiectul a fost organizat in trei pachete principale : GUI , Polinom si TestareUnitara

In Pachetul Polinom se gasesc clasele Polinom si Monom care se ocupa de procesarea polinomului respectiv a monoamelor

Dupa ce utilizatorul introduce polinomul in campul TextField din GUI se construieste in memorie un nou polinom . Polinomul in TextField este reprezentat ca si un String . In constructorul clase Polinom se apeleaza metoda readPolinom(String n) prin care se descompune polinomul in mai multe monoame.

```
private void readPolinom(String n){//metoda prin care se citeste polinomul deja verificat
  this.polinom.removeAll(polinom);
  String[] f= new String[n.length()];//dupa metoda de verificare aici vor fi stocati termeni polinomului
  if(!vfReg(n,f)||n.equals("0"))return; // daca metoda de verificare nu este buna se anuleaza procesarea
  int i=0;
  double coef=1,exp=0;
  //in ciclul while se despar monoamele in coeficienti si exponenti
  while(i<f.length && f[i]!=null){
    if(f[i].indexOf('x')==f[i].length()-1){
      exp=1;
      if(f[i].equals("x")) coef = 1;
      else if(f[i].equals("-x")) coef = -1;
      else coef = Double.parseDouble(f[i].substring(0,f[i].indexOf('x')));
    }else if(f[i].indexOf('x')!=-1){
      if(f[i].indexOf('x')==0)coef=1;
      else if(f[i].indexOf('x')==1 &&f[i].indexOf('-')==0)coef=-1;
      else coef = Double.parseDouble(f[i].substring(0,f[i].indexOf('x')));
      exp = Double.parseDouble(f[i].substring(f[i].indexOf('x')+1,f[i].length()));
    }else if(f[i].indexOf('x')==-1){
      coef = Double.parseDouble(f[i]);
      exp = 0;
    }
    i++;
    this.polinom.add(new Monom(coef,exp));//se adauga un nou monom dupa ce sau procesat
    //coeficienti si exponentii
  }
  Collections.sort(this.polinom);//ca polinomul sa arate frumos se sorteaza
}
```

Metoda readPolinom selecteaza coeficienti si exponenti din polinomul introdus de utilizator dar nu inainte sa fie verificat de metoda vfReg(String n,String[] x) prin care se indentifica daca polinomul contine expresi ilegale .

Metoda vfReg(String n, String[] x) are doi parametri String n si String[] x . Parametrul n este String-ul introdus de utilizator si x este un tablou de String-uri care va fi refolosit si in readPolinom deoarece acesta va retine toate monoamele din polinom .

```
private boolean vfReg(String n,String[] x){

  if(n.isEmpty())return false;

  String myStr= n.replace(" ","");// se elimina spatiile

  String reg = "(^([-]?((\\d{0,100})|(\\d{1,100}([*]?)))[xX][\\\\^]\\d{1,100})|(^([-
+]?)((\\d{0,100})|(\\d{1,100}([*]?)))[xX])|(^[-+]?\\d{1,100}))" +

      "|([-+]((\\d{0,100})|(\\d{1,100}([*]?)))[xX][\\\\^]\\d{1,100})|([-
+]((\\d{0,100})|(\\d{1,100}([*]?)))[xX])|([-+])\\d{1,100}";//monom bun

  //se foloseste RegEx pentru verificarea Monoamelor si indentificarea Expresilor ilegale

  Pattern pattern = Pattern.compile(reg);

  Matcher m = pattern.matcher(myStr);

  StringBuilder f= new StringBuilder();

  //variabila f este un String prin care se vor pune la loc toate Monoamele si se verifica daca expresia e
legala

  int i=0;

  while(m.find()) {

    String f1=m.group();

    f1=f1.replaceAll("[\\\\^+*]","").replace("X","x");

    //se elimina din group toate semnele pentru o usoara citire a numerelor si se inlocuieste X cu x

    x[i++]=f1;//Se adauga in tablou String-ul procesat

    f.append(m.group());

  }

  return f.toString().equals(myStr);//returneaza fals in cazul in care expresia este invalida

}
```

Astfel dupa ce polinomul a fost descifrat prin aceste doua metode el urmeaza sa fie folosit in anumite operatii . Rezultatul va fi afisat in TextArea din GUI sub forma de String . Ca acest lucru sa fie posibil am folosit metoda afisPolinom() care ia fiecare Monom si il reconstruieste ca String . Monomul la randul lui trebuie reconstruit de acea in clasa Monom exista metoda afisMonom(). Metoda afisMonom() este descrisa mai jos , la partea in care am prezentat metodele din Monom.

```
public String afisPolinom(){//se formeaza un String din monoamele avute in polinom

  String rez="";

  for(Monom m :polinom){

    if(polinom.get(0).equals(m))rez+=m.afisMonom().replace("+","");

      //in cazul in care suntem la primul element ,se renunta la semn daca ceoficientul este pozitiv

    else rez+=m.afisMonom();

  }

  return (this.polinom.size()==0)?"0":rez;

}
```

Adunare

Operatia de Adunarea este reprezentata prin metoda adunare(Polinom p) care contine parametrul p care este de tip Polinom . p reprezinta polinomul cu care se va aduna la polinomul current “this” . Metoda returneaza un nou polinom care contine rezultatul acestei operatii.

```
public Polinom adunare(Polinom p){

  Polinom rezultat = new Polinom();

  this.reducere();

  p.reducere();

  rezultat.coppyArray(p.polinom);//se adauga toti termeni din p in rezultat pentru a evita uitarea unor
termeni

  boolean ok ;

  for(Monom m : this.polinom){

    ok = true;

    for(Monom x : rezultat.polinom)

      if(m.getExp()==x.getExp()) {
```

```
       x.setCoef(x.getCoef()+m.getCoef());

       ok=false;

     }

   if(ok)rezultat.polinom.add(new Monom(m.getCoef(), m.getExp()));//in cazul in care un termen exista dar
nu poate fi

   //adunat se adauga in noul polinom

  }

  rezultat.reducere();

  return rezultat;

}
```

Dar inainte de returnarea rezultatului apare metoda reducere() . Aceasta metoda reduce termeni asemenea .

```
public void reducere(){//reduce termeni asemenea din polinom

  Collections.sort(this.polinom);

  Polinom s=new Polinom();

  Monom f = new Monom(0,0);

  for(Monom m :this.polinom){

   if(m.getExp()==f.getExp())f.setCoef(m.getCoef()+f.getCoef());//se iau termeni cu aceasi exponent si se
aduna

   else {

     if(f.getCoef()!=0) s.polinom.add(f);
```

Ex: 2X-2X = 0 ; 2X-3X = -X

Scadere

Operatia de Scadere este reprezentata prin metoda scadere(Polinom p) care contine parametrul p care este de tip Polinom . p reprezinta polinomul scazotor din care descazutul “this” este scazut . Metoda returneaza un nou polinom care contine rezultatul acestei operatii.

```
public Polinom scadere(Polinom p){//pe acelasi principiu cu adunarea doar ca aici coeficienti sunt cu -
  Polinom rezultat = new Polinom();
  this.reducere();
  p.reducere();
  rezultat.coppyArray(this.polinom);
  boolean ok ;
  for(Monom m : p.polinom){
    ok = true;
    for(Monom x : rezultat.polinom)
      if(m.getExp()==x.getExp()) {
        x.setCoef(x.getCoef()-m.getCoef());// Atentie: this - p NU: p - this
        ok=false;
      }
    if(ok)rezultat.polinom.add(new Monom(-m.getCoef(), m.getExp()));
  }
  rezultat.reducere();
  return rezultat;
}
```

Inmultire

Operatia de Inmultire este reprezentata prin metoda imultire(Polinom p) care contine parametrul p care este de tip Polinom .Metoda foloseste parametrul p si parcurge fiecare element prin doua for-uri si inmulteste monom cu monom fara sa tina cont de exponent. Metoda returneaza un nou polinom care contine rezultatul acestei operatii.

```
public Polinom inmultire(Polinom p){
  Polinom s = new Polinom();
  this.reducere();
  p.reducere();
  for(Monom m : this.polinom){
    for(Monom x : p.polinom)
```

```
      s.polinom.add(new Monom(m.getCoef()*x.getCoef(),m.getExp()+x.getExp()));

  }//in aceste for-uri se inmultesc toti coeficienti intre cei doi polinomi si se aduna exponenti

  Polinom c = new Polinom();

  c.coppyArray(s.polinom);

  for(Monom m : c.polinom)if(m.getCoef()==0)s.polinom.remove(m);//este necesara eliminarea elementelor
nule

  s.polinom=c.polinom;

  s.reducere();

  // cu toate ca sau eliminat elementele nule este necesara o reducere pentru adunarea elementelor de
acelas grad

  return s;

}
```

Impartire

Operatia de Impartire este reprezentata prin metoda impartire(Polinom impartitor) care contine parametrul impartitor care este de tip Polinom . Algoritmul folosit este cel care a fost predat in liceu . Metoda este compara gradele celor doua polinoame si daca gradul deimpartitului este mai mare decat gradul impartitorului se efectueaza operatia , in caz contrar se returneza ca rest deimpartitul .Am creat un Polinom auxiliar “aux” care va fi folosit sa retina doar un singur monom , acest monom fiind monomul cu care se efectueaza operatia de inmultire cu impartitorul si operatia de scadere cu deimpartitul. Deimpartitul se actualizeaza si se compara din nou gradul cu cel al impartitorului . Metoda returneaza un ArrayList<Polinom> de polinoame . Primul polinom reprezinta catul si cel de al doilea polinom reprezinta rest.

```
public ArrayList impartire(Polinom impartitor){

  this.reducere();

  impartitor.reducere();

  if(impartitor.polinom.size()==0||this.polinom.size()==0)return new ArrayList(0);//in caz ca polinomul e
invalid se returneaza 0

  double exp2=impartitor.polinom.get(0).getExp() , exp1=this.polinom.get(0).getExp();

  double coef2=impartitor.polinom.get(0).getCoef() , coef1=this.polinom.get(0).getCoef();

  Polinom deimpartit= new Polinom();

  Polinom aux= new Polinom();
```

```
  deimpartit.coppyArray(this.polinom);

  Polinom cat= new Polinom();

  while (exp1>=exp2){//in caz ca exponentul deimpartitului este mai mare sau egal cu cel al impartitorului
se continua algoritmul

    cat.polinom.add(new Monom(coef1/coef2,exp1-exp2));//se adauga in Cat deimpartit/impartitor

    aux.polinom.add(cat.polinom.get(cat.polinom.size()-1));//in polinomul auxiliar se adauga monomul cu

    //grad minim pentru calcularea restului

    aux=aux.inmultire(impartitor);

    deimpartit=deimpartit.scadere(aux);//se scade din deimpartit (monomul cu grad minim)*impartitor

    Collections.sort(deimpartit.polinom);//sorteaza astfel incat sa se aduca la forma normala a unui
polinom

    Collections.sort(impartitor.polinom);

    if(deimpartit.polinom.size()==0)break;//ca sa nu se produca vreo eroarea la asignari se intrerupe
while-ul

    exp2=impartitor.polinom.get(0).getExp();

    exp1=deimpartit.polinom.get(0).getExp();

    coef2=impartitor.polinom.get(0).getCoef();

    coef1=deimpartit.polinom.get(0).getCoef();

    aux.polinom.removeAll(aux.polinom);// in aux ne intereseaza doar ultimul element din Cat

  }

  ArrayList<Polinom> n = new ArrayList<>();//construieste Catul si Restul

  n.add(cat);

  n.add(deimpartit);//deimpartitul ramas devine restul impartiri

  return n;
```

Derivare

Derivarea polinoamelor se face cu ajutorul metodei derivPolinom un ciclu foreach prin care se parcurg fiecare din monoamele polinomului si se aplica metoda derivMonom prin care este posibila derivarea. Derivarea se face prin scaderea exponentului si inmultirea coeficientului cu fostul exponent.Metoda returneaza un nou polinom cu rezultatul dorit.

```
public Polinom derivPolinom(){//deriveaza dupa fiecare monom

  Polinom p = new Polinom();
```

```
  this.reducere();

  p.coppyArray(this.polinom);

  for(Monom m : p.polinom)m.derivMonom();

  p.reducere();//necesar pentru eliminarea zerourilor extra

  return p;

}
```

Integrare

Integrarea polinoamelor se face cu ajutorul metodei integPolinom un ciclu foreach prin care se parcurg fiecare din monoamele polinomului si se aplica metoda integMonom prin care este posibila integrarea. Integrarea se face prin incrementarea exponentului si impartirea coeficientului cu exponentul.Metoda returneaza un nou polinom cu rezultatul dorit.

```
public Polinom integPolinom(){//integreaza dupa fiecare monom

  Polinom p = new Polinom();

  this.reducere();

  p.coppyArray(this.polinom);

  for(Monom m : p.polinom)m.integMonom();

  return p;

}
```

In clasa polinom se mai afla si functia private coppyArray care face deep coppy unui polinom si metoda getPolinom care imi returneaza lista cu monoame.

```
private void coppyArray(List<Monom> x){//copiaza element cu element fiecare Monom din x in this

  this.polinom.removeAll(this.polinom);//elementele de care nu mai este nevoie se sterg

  for(Monom m : x)this.polinom.add(new Monom(m.getCoef(),m.getExp()));

}
```

Monom

In clasa Monom se afla get-erele si set-erele folosite pentru a returna atributele de clasa si metodele de derivare ,integrare, convertire la String , de setare

a precizie pentru afisarea numarului minim de zecimale pentru fiecare coeficient si metoda compareTo.

Clasa Monom implementeaza interfata Comparable pentru sortarea elementelor polinomului in functie de grad (exponent) . Este necesara sortarea polinomului pentru ca utilizatorul sa primeasca polinomul dupa forma normala X^n +…X^0

Metoda variabilePrecision(double x) are ca parametru x si returneaza numarul minim de zecimale necesare pentru afisarea polinomului in forma cea mai “User Friendly” din punct de vedere a aspectului. Metoda functioneaza prin convertirea la long a variabilei x si inmultirea ei cu 10 pentru a afla prima zecimala diferita de 0 . Toate acestea fiind intr-un ciclu while. Metoda returneaza un int acesta reprezentand numarul de zecimale . Se returneaza minim doua zecimale.

```
private int variablePrecision(double x){//determina precizia lui x
  int precision=1;
  if(x%1==0)return 1;//in cazul in care x este 0 se retuneaz precizia 1 . Adica are doar o zecimala
  x=(x%1);
  while(true){
    if((long)(x*=10)!=0)break;//se verifica cand ajunge x sa aiba macar o zecimala diferita de 0
    precision++;
  }
  return precision+1;//precizia minima este 2
}
```

Metoda cea mai importanta din aceasta clasa este afisMonom() prin care se reconstruieste monomul sub forma de String . Monomul este format din exponent si din coeficient , acestea sunt si atributele pe care clasa le retine . In functie de exponent putem avea afisari diferite . Adica daca epxonentul nostru este egal cu 1 nu se va mai scrie x^1 ci se va reduce la forma x . Cu toate acestea , aceasta metoda nu este de ajuns deoarece se poate intampla sa avem un numar cu coeficient mai mic ca 0 cea ce inseamna ca trebuie facuta schimbare de semn. Mai este cazul special care trebui tratat acesta fiind ax^0 . In acest caz programul nu va

mai afisa ridicarea la putere , va afisa doar a , adica coeficientul lui x^0 . Am observat ca se poate intampla , in cazul impartiri si a integrari ca acesti coeficienti sa nu mai fie intregi , din aceasta cauza am tratat si cazul in care coeficentul este intre -1 si 1 .

```
public String afisMonom(){//functia de construire a String-ului pentru monom

  long x= (long) coef, y = (long) exp;//in cazul in care coef si exp sunt intregi nu se vor afisa cu
zecimale

  String e=String.valueOf(y);

  String c;

  if(coef%1==0.0f)c=String.valueOf(x);

  else c=String.format("%."+variablePrecision(coef)+"f",coef);//am construit o functie care determina
precizia

  //cu care se pot afisa elementele in cazul in care sunt cu virgula

  if(exp==1)return (coef>1)?"+"+c+"x":((coef<-1)?c+"x":((coef==-1)?"-
x":((coef<1&&coef>0)?"+"+c+"x":((coef>-1&&coef<0)?c+"x":"+x"))));

  // pentru exp =1 se va afisa doar x . NU x^1 . Am determinat String-ul pentru fiecare coef

  else if(exp==0)return (coef>0)?"+"+c:c;//pentru x^0 se afiseaza doar coeficientul

  else if(coef<1&&coef>-1) return (coef<1&&coef>0)?"+"+c+"x^"+e:c+"x^"+e;//pentru 0.xx se adauga [+-
]0.xxX^e

  else return (coef==1)?"+x^"+e:((coef==-1)?"-x^"+e:((coef>1)?"+"+c+"x^"+e:c+"x^"+e));

  // cand coeficientul este 1 se va afisa doar x .NU 1x sau 1*x

}
```

GUI

Pentru partea grafica am folosit framework-ul JavaFX . Am putut crea cu aceste librarii interfata grafica si graficul functiilor pe care utilizatorul le introduce in TextField-uri .

In pachetul GUI se afla clasa Interfata .Clasa prezinta multe atribute de clasa acestea fiind private. Am ales acest mod de lucru deoarece imi permite mai multa flexibilitate . Ca atribute de clasa avem : adunare, scadere, inmultire, impartire ,derivare , integrare si graph acestea reprezentand butoanele pe care utilizatorul interactioneaza cu interfata grafica . TexField-urile xRead si yRead reprezinta zona de text in care utilizatorul introduce polinoamele sub forma de String , acestea sunt

tot atribute de clasa . Atributul care ne arata rezultatul se numeste result si este un TextArea . Atributele pentru grafic sunt xAxis si yAxis care sunte de tipul NumberAxis , acestea reprezinta axele graficului . Graficul propiuzis este de tipul LineChart si are ca parametri doua numere . Ultimi parametri sunt seriesX si seriesY de tipul XYChart.Series acestea reprezinta graficul celor doua polinoame. Am ales sa folosesc doar o singura clasa pentru interfata grafica deoarece in JavaFX este mult mai usor de lucrat asa. Interfata prezinta urmatoarele metode :

-start(Stage stage): aceasta metoda initializeaza Stage-ul creaza panel-urile si proiecteaza Scene-ul . In Scene sunt adaugate in ordine toate panel-urile create si la final Scene-ul se adauga in Stage. Inainte de terminarea metodei se apeleaza metoda stage.show() pentru ca interfata sa fie vizibila utilizatorului.

-initValues() : este metoda prin care se initializeaza variabilele de clasa . Este o metoda facuta private deoarece poate fi apelata o singura data in metoda start(…) fara sa poata sa fie folosita in alta parte.

-initGraph() : este metoda prin care se initializeaza si pregateste partea de grafic pentru functii. Aceasta construieste axa numerelor si serile de valori pentru afisarea lor in graf (NumberAxis ,XYChart.Series).

-handle (ActionEvent event): este metoda prin care se creaza interactiune utilizatorului cu butoanele din interfata . Utilizatorul cand apasa pe butonul de adunare , de exemplu , se va apela metoda de adunare pentru cele doua polinoame

-scaleGraph(Polinom f) : este metoda prin care se aduce pe grafic functia gata calculata astfel incat sa incapa pe grafic . Metoda are nevoie de limitele setate de grafic asa ca apeleaza inauntrul ei metoda getUpperBounds() si getLowerBounds() pentru o scalare corecta

-graphBounds(Polinom f , Polinom s) : este metoda care seteaza limitele graficului in functie de polinoamele introduse de utilizator . Deoarece se pot interpreta doua polinoame pe acelasi grafic , trebuie sa se ia limitele celui mai mare polinom in functie de ultimul coeficient. Metoda nu este cea mai eficienta pentru adaugare graficului dar prezinta siguranta

-showGraph(): este metoda care ne arata apeleaza functia de removeAll din grafic si care urmeaza s ail rescrie prin metodele scaleGraph(…) si graphBounds(…).

-main(String[] args): aici se executa tot programul . In metoda main am apelat functia launch(args) din clasa Application astfel incat programul sa ruleze.

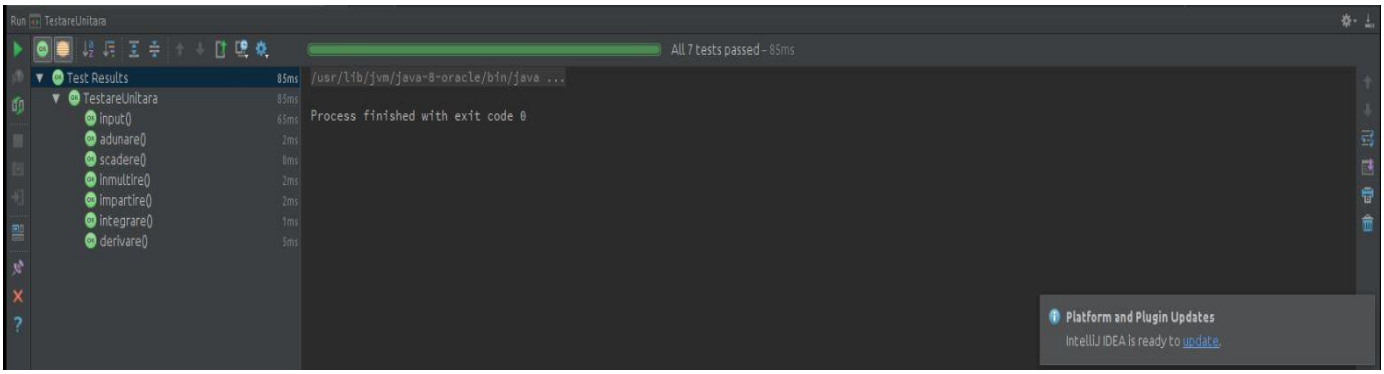
5. Testare

Pentru TestareUnitara sa folosit JUnit si am apelat testarea pe urmatoarele metode : adunare(…) ,scadere(…) , inmultire(…) , impartire(…) , derivare() , integrare() si pe metoda de verificare a input-ului .

```
public class TestareUnitara {

  private Polinom x ;

  private Polinom y ;

  @Test

  public void input(){

    x=new Polinom("X^2+1");

    assertEquals(x.afisPolinom(),"x^2+1");

    x=new Polinom("X^x+1");

    assertEquals(x.afisPolinom(),"0");

    x=new Polinom("0");

    assertEquals(x.afisPolinom(),"0");

    x=new Polinom("X");

    assertEquals(x.afisPolinom(),"x");

    x=new Polinom("2x+asd");

    assertEquals(x.afisPolinom(),"0");

    x=new Polinom("2*x^2+3*x");

    assertEquals(x.afisPolinom(),"2x^2+3x");

  }

  @Test
```

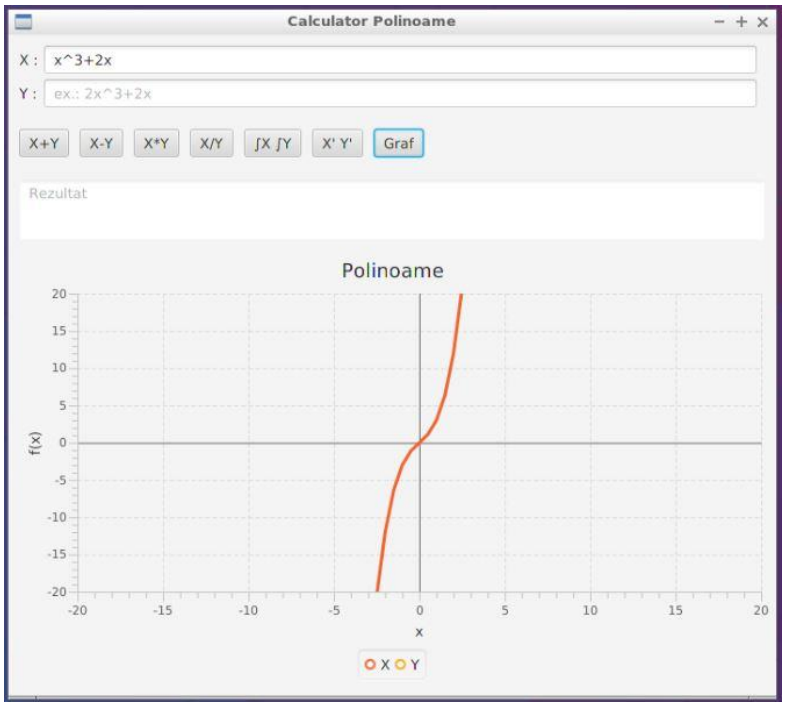
```
public void adunare(){
  x=new Polinom("X^2+1");
  y=new Polinom("X^2+1");
  assertEquals(x.adunare(y).afisPolinom(),"2x^2+2");
}
@Test
public void scadere(){
  x=new Polinom("2X^2+1");
  y=new Polinom("X^2+1");
  assertEquals(x.scadere(y).afisPolinom(),"x^2");
}
@Test
public void inmultire(){
  x=new Polinom("X+1");
  y=new Polinom("X+1");
  assertEquals(x.inmultire(y).afisPolinom(),"x^2+2x+1");
}
@Test
public void impartire(){
  x=new Polinom("X^2+2x+1");
  y=new Polinom("X+1");
  ArrayList<Polinom> a=x.impartire(y);
  String s=a.get(0).afisPolinom()+" "+a.get(1).afisPolinom();
  assertEquals(s,"x+1 0");
}
@Test
public void integrare(){
  x=new Polinom("2x+1");
  assertEquals(x.integPolinom().afisPolinom(),"x^2+x");
}
@Test
```

```
  public void derivare(){

    x=new Polinom("2x+1");

    assertEquals(x.derivPolinom().afisPolinom(),"2");

  }
}
```

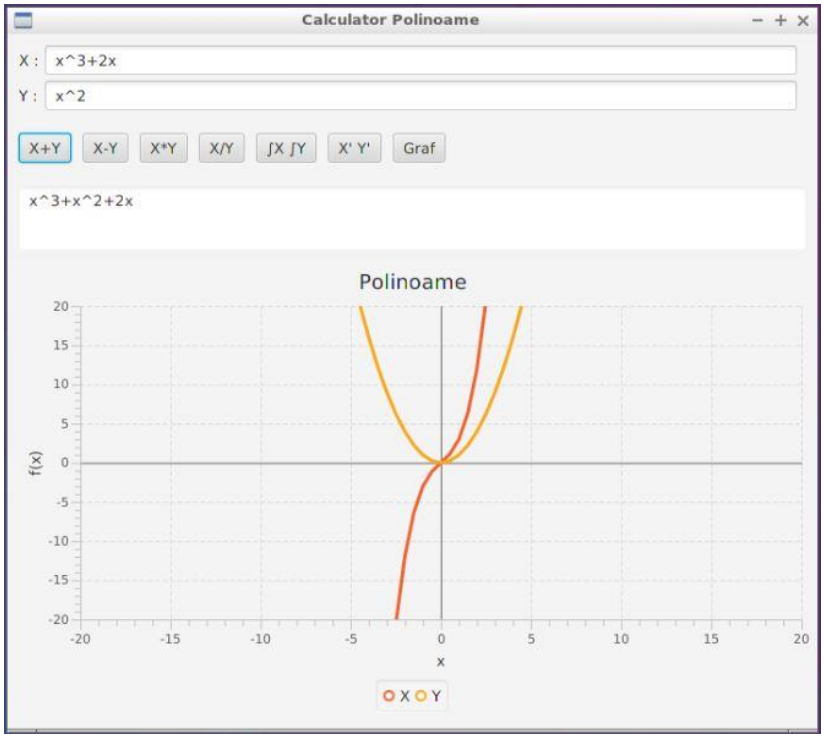


6. Rezultate

Grafic :

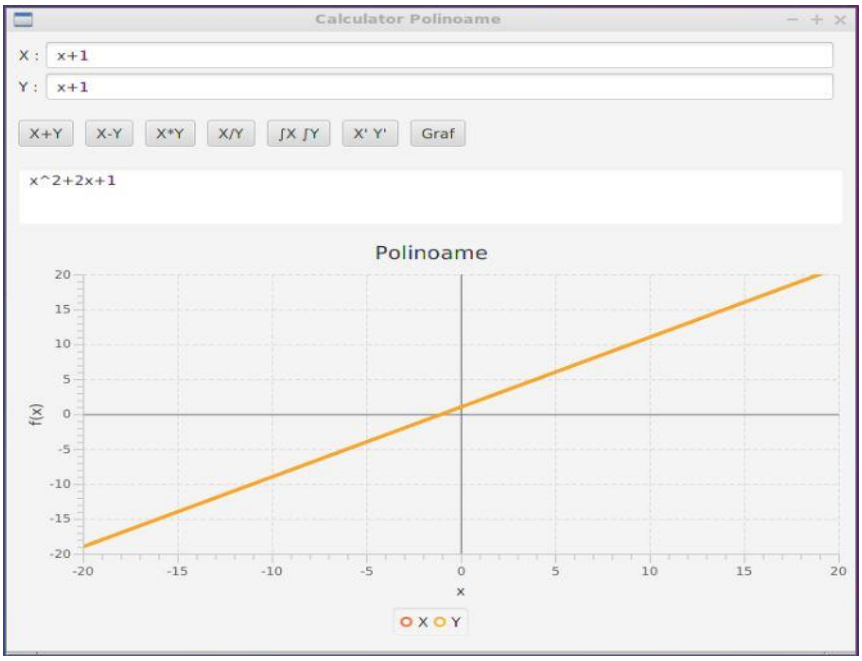


Adunare :

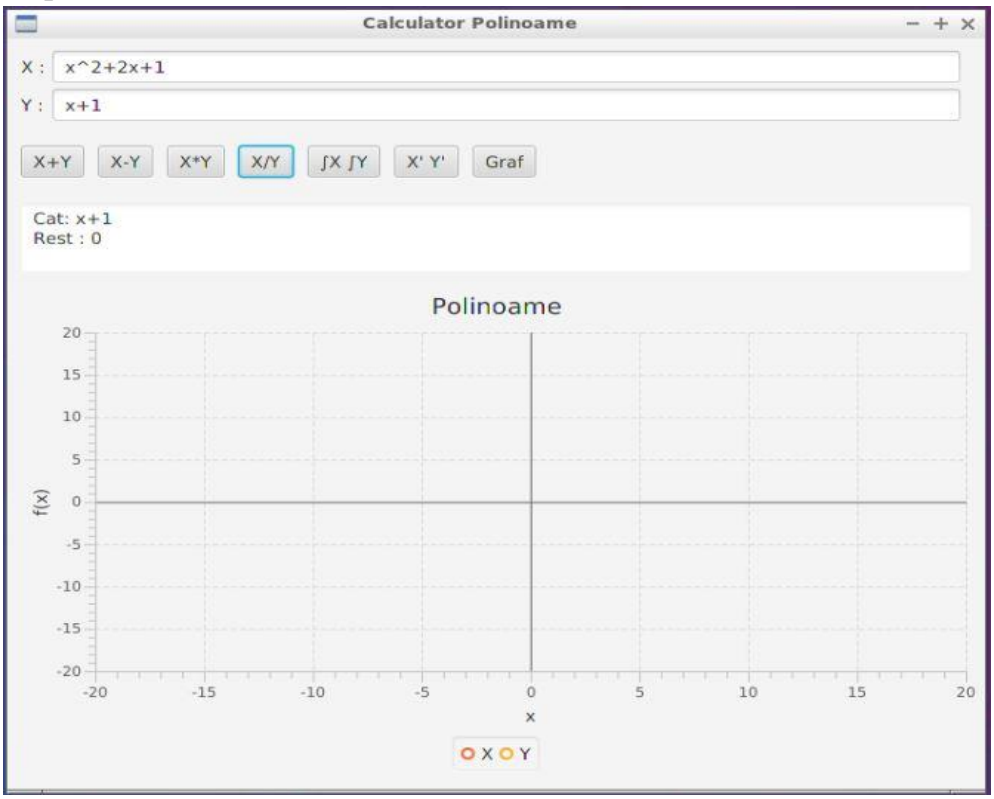


Scadere :

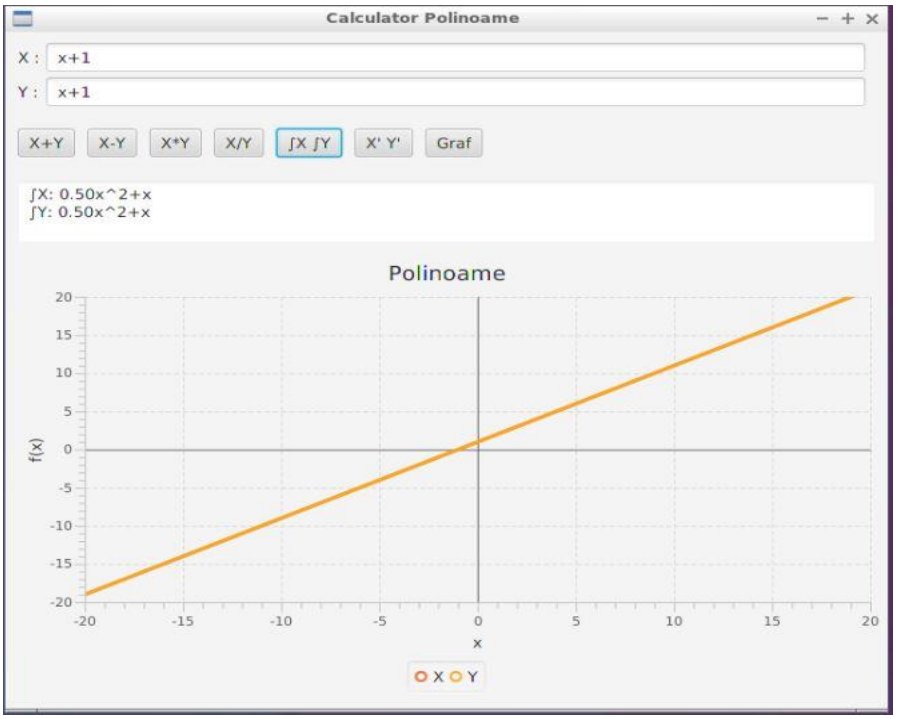
Inmultire :



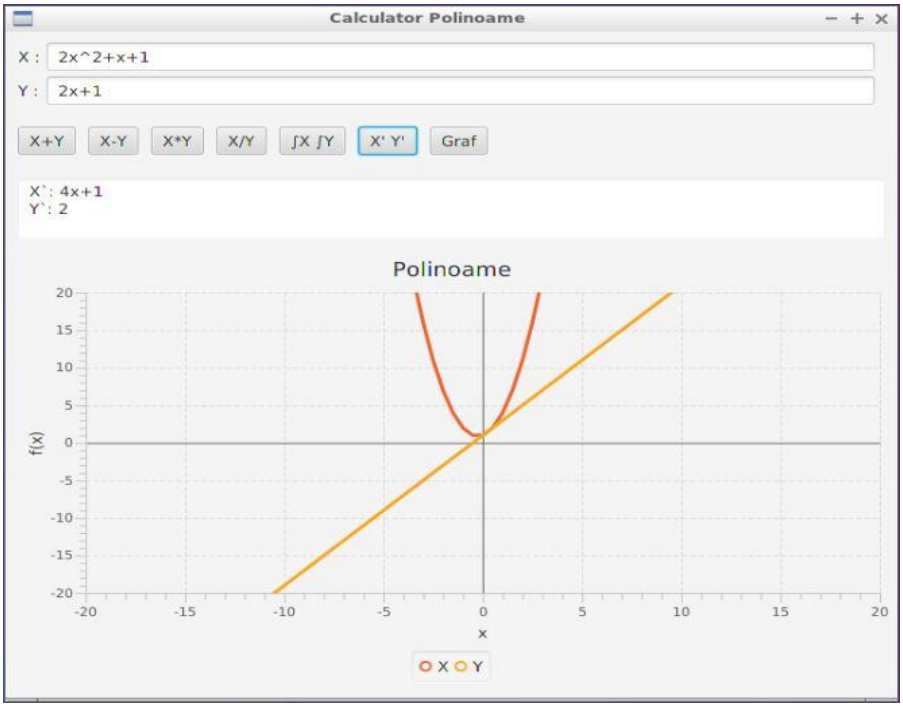
Impartire:



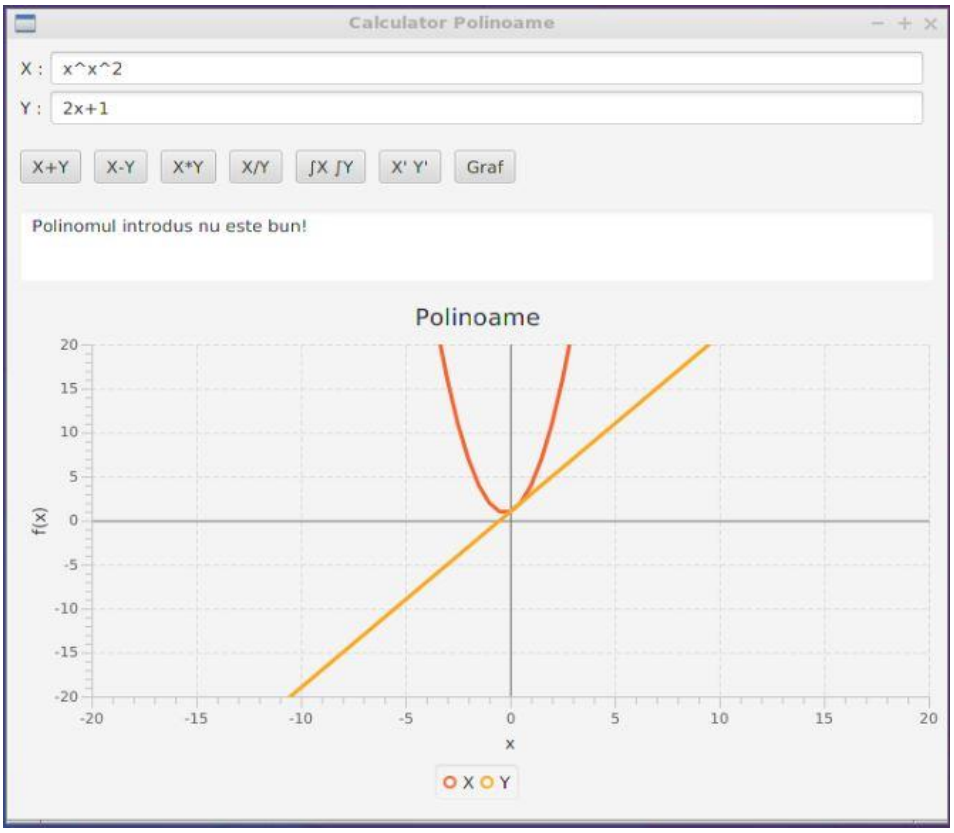
Integrare:



Derivare :



Eroare :



7. Concluzii si Dezvoltari Ulterioare

Cu acest proiect pot sa spun ca mi-am intarit cunostintele de OOP si ca am invatat sa programez cu usurinta o interfata grafica . Din punctul meu de vedere proiectul ma facut sa respect mai mult munca de programator si ma facut si mai interesat de acest domeniu . Acest program se poate folosi de profesori pentru predarea polinoamelor pentru elevi .

Pentru acest proiect se poate aduce o optimizare in cea ce priveste algoritmul pentru trasarea graficelor plus o functie prin care se creaza o scalare mai precisa, prin aflarea radacinilor si calcularea graficului prin derivate . O metoda care sa stabileasca punctele de intersectie intre cele doua grafice si care sa poata calcula asimptotele pentru grafic , aceasta ar fi o imbunatatire . O alta imbunatatire ar fi aducerea unei metode care iti poate ridica tot polinomul la o anumita putere .

8. Bibliografie

https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm

https://lankydanblog.com/2017/01/29/javafx-graphs-look-pretty-good/

https://en.wikipedia.org/wiki/Regular_expression

http://www.regular-expressions.info/tutorial.html