Calculator de polinoame

**1.Obiectivul temei**

Obiectivul principal al temei este proiectarea si implementarea unui calculator de polinoame cu o interfata grafica dedicata prin care utilizatorul sa introduca doua polinoame si sa efectueze operatiile de adunare, scadere, inmultire si impartire pe acestea. Pe langa operatiile aritmetice amintite trebuie implementate si operatiile de derivare si integrare a unui polinom.

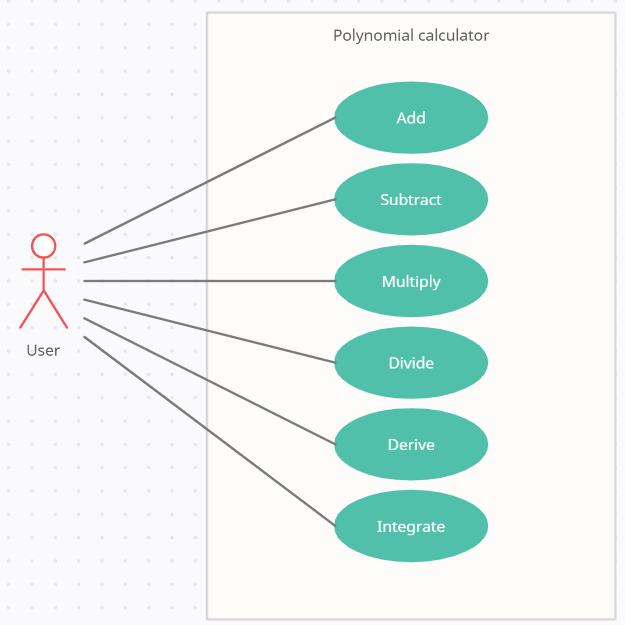
Sub-obiective:

* Analiza problemei si identificarea necesitatilor
* Proiectarea calculatorului de polinoame
* Implementarea calculatorului de polinoame
* Testarea calculatorului de polinoame

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

Principalele proprietati ale aplicatiei sunt:

* Singurul actor este utilizatorul
* Acesta beneficiaza de functionalitatile calculatorului de polinoame prin interfata grafica dedicata
* Utilizatorul va putea introduce de la tastatura unul sau doua polinoame pe care se vor efectua una dintre operatiile expuse in diagrama use-case.
* Fiecare operatie va avea un buton aferent
* Rezultatul va fi afisat imediat dupa apasarea butonului



In primul rand este de dorit ca aplicatia sa ofere o interfata grafica simplista si intuitiva care sa nu lase nici o urma de echivoc in folosirea acesteia de catre utilizator. Astfel se elimina orice necesitate de cunostinte tehnice prealabile legate de aplicatie, iar spectrul diversitatii de posibili utilizatori se mareste considerabil.

In al doilea rand, pentru aceasta aplicatie am ales o implementare graduala, bottom-up. Cu alte cuvinte voi incepe prin implementarea necesitatilor de baza pe care mai apoi se vor cladi toate functionalitatile aplicatie.

Caz de utilizare: add / subtract / multiply / divide

Actor principal: utilizatorul

Scenariu:

1. Utilizatorul introduce doua polinoame in interfata grafica
2. Utilizatorul selecteaza operatia dorita (aduna<add>, scade<subtract>, inmulteste<multiply>, imparte<division>) apasand pe butonul aferent
3. Calculatorul de polinoame realizeaza adunarea celor doua polinoame si afiseaza rezultatul in interfata grafica

Caz de utilizare: derive / integrate

Actor principal: utilizatorul

Scenariu:

1. Utilizatorul introduce un polinom (Polynomial 1) in interfata grafica
2. Utilizatorul selecteaza operatia dorita (deriveaza <derive>, integreaza <integrate>) apasand pe butonul aferent
3. Calculatorul de polinoame realizeaza derivarea sau integrarea polinomului si afiseaza rezultatul in interfata grafica

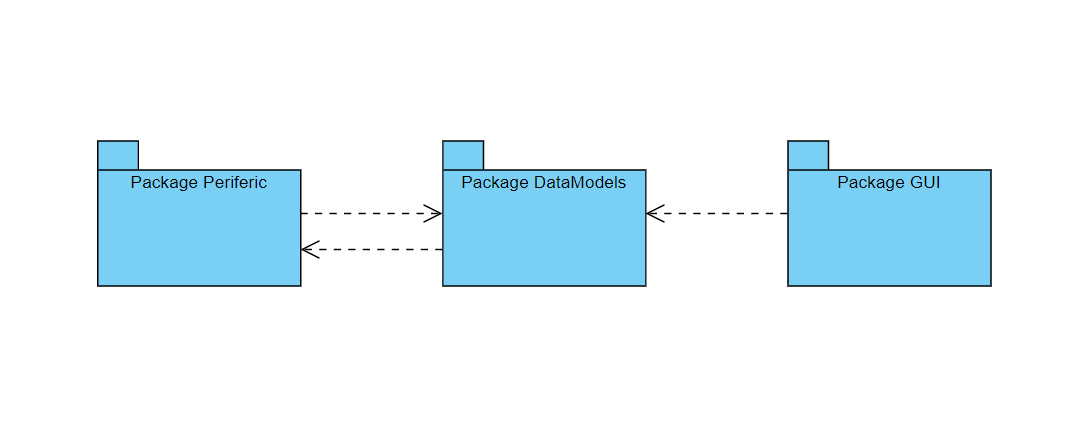
**3.Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, algoritmi, interfata utilizator)**

Voi imparti proiectul in trei pachete: DataModels, GUI, Periferic.

In pachetul DataModels se vor crea toate functionalitatile necesare operatiilor pe polinoame.

In pachetul GUI se va realiza interfata grafica care va folosi functionalitatile din DataModel.

In pachetul Periferic se vor realiza anumine functionalitati secundare.



Un polinom este format din mai multe monoame. Din aceasta cauza polinomul va fi implementat ca o lista de monoame folosind ArrayList. Astfel clasa monoamelor va fi formata din doua atribute necesare si suficiente pentru a fi reprezentate: degree (gradul monomului), coefficient (coeficientul monomului).

Partea de interfata grafica am ales sa o realizez in JavaFX si cuprinde doua campuri in care utilizatorul va introduce polinoamele asupra carora se vor efectua operatiile dorite(polinom\_1, polinom\_2), un camp unde va fi afisat rezultatul(polinom\_rez) si sase butoane de unde se va selecta operatia(Add, Subtract, Multiply, Divide, Derive, Integrate).

Pentru a desparti String-ul citit de la tastatura in String-uri de monoame voi folosi Regular Expression (Regex) pentru ca mai apoi sa identific coeficientul si exponentul monomului si sa-l creez ca instantiere a clasei monoamelor.

Polinoamele se vor introduce de la tastatura fara spatii intre ele si folosind caracterul ’^’ ca notatie a ridicarii la putere. Se pot introduce polinoame cu grade si coeficienti de orice dimensiuni si in orice ordine.

EX: 3x^3+2x+1, x^10+x, -x^2+x^9-1

Pentru aceasta aplicatie voi implementa 6 clase principale: PolynomialWithIntegerCoefficient, MonomialWithIntegerCoefficient, PolynomialWithRealCoefficient, MonomialWithRealCoefficient, Controller si sample.fxml (view).

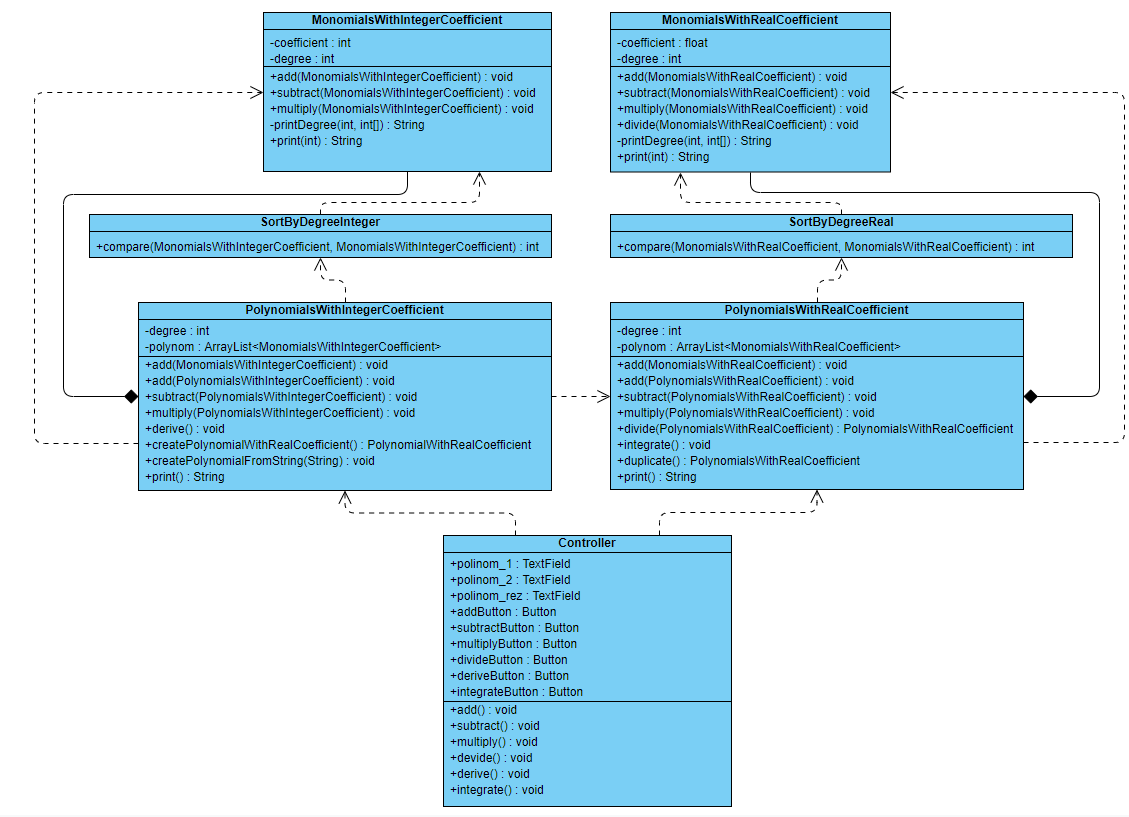
Clasele PolynomialWithIntegerCoefficient si MonomialWithIntegerCoefficient le voi folosi pentru implementarea operatiilor de adunare, scadere, inmultire si derivare, iar clasele PolynomialWithRealCoefficient si MonomialWithRealCoefficient pentru operatiile de integrare si impartire(unde polinomul rezultant poate avea coeficienti reali).

Practic voi incepe prin realizarea operatiilor care nu necesita polinoame cu coeficienti reali. Pentru acestea, deoarece adunarea, scaderea si inmultirea polinoamelor reprezinta o serie de mai multe adunari, scaderi si inmultiri de monoame, am implementat metode aferente in clasa MonomialWithIntegerCoefficient.

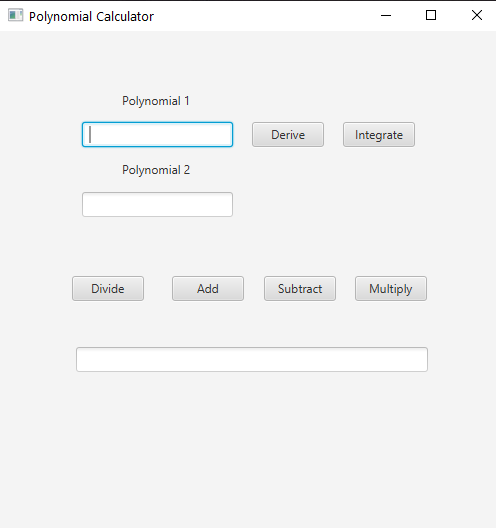
Dupa aceea voi implementa clasele si metodele necesare pentru a realiza operaratiile de impartire si integrare.

Clasa Controller va realiza legatura dintre interfata grafica (sample.fxml) si pachetul DataModels. Prin aceasta se vor putea apela toate metodele necesare pentru lucrul cu polinoame.

Diagrama UML pentru clase:



Pentru GUI am creat doua TextField-uri (polinom\_1, polinom\_2) in care utilizatorul va putea introduce polinoamele asupra carora se vor efectua operatiile implementate, sase butoane pentru alegerea operatiei dorite( addButton, subtractButton, multiplyButton, divideButton, deriveButton, integrateButton) si inca un TextField (polinom\_rez) pentru afisarea rezultatului.



**4. Implementare**

***Clasa MonomialWithIntegerCoefficient:***

Ca atribute, aceasta clasa contine gradul monomului si coeficientul acestuia, ambele private.

private int degree;  
 private int coefficient;

Metode principale din aceasta clasa sunt cele de adunare, scadere si inmultire de monoame care vor fi folosite la adunarea, scaderea si inmultirea polinoamelor:

public void add(MonomialWithIntegerCoefficient a) {  
 if(a.degree!=this.degree)  
 System.*out*.println("Error: You are trying to add monomials(integer coefficient) with different degrees!");  
 else this.coefficient += a.coefficient;  
 }  
  
 public void subtract(MonomialWithIntegerCoefficient a) {  
 if(a.degree!=this.degree)  
 System.*out*.println("Error: You are trying to subtract monomials(integer coefficient) with different degrees!");  
 else this.coefficient -= a.coefficient;  
 }  
  
 public void multiply(MonomialWithIntegerCoefficient a) {  
 this.coefficient \*= a.coefficient;  
 this.degree = this.degree + a.degree;  
 }

In aceste metode, pentru a ajuta la debugging, se va afisa in consola mesaje de eroare in cazul in care se incearca adunarea sau scaderea de monoame cu grade diferite.

Pe langa acestea am realizat si doua metode necesare pentru afisarea monoamelor (care, din nou, vor fi folosite pentru afisarea polinomului rezultat in urma operatiilor):

private String printDegree(int a, int []p) {  
 String s=new String();  
 while(a!=0) {  
 s=s+Character.*toString*((char) p[a % 10]);  
 a/=10;  
 }  
 String r=new String();  
 for(int i=s.length()-1; i>=0; i--)  
 r=r+s.charAt(i);  
 return r;  
 }  
  
 public String print(int c) {  
 String s=new String();  
 int []p={8304, 185, 178, 179, 8308, 8309, 8310, 8311, 8312, 8313};  
 if(c!=0 && this.coefficient>0)  
 s=s+'+';  
 if(this.coefficient==1) {  
 if(this.degree==0)  
 s=s+this.coefficient;  
 }  
 else if(this.coefficient==-1) {  
 if(this.degree==0)  
 s=s+this.coefficient;  
 else  
 s=s+'-';  
 }  
 else  
 s=s+this.coefficient;  
 if(this.degree!=0)  
 s=s+'x';  
 int a=this.degree;  
 if(a>1) s=s+printDegree(a, p);  
 return s;  
 }

Metoda *print* returneaza monomul ca String si foloseste metoda printDegree() pentru o afisare cat mai user-friendly.

***Clasa PolynomialWithIntegerCoefficient:***

Ca atribute, aceasta clasa contine gradul polinomului si un ArrayList de monoame cu coeficient intreg, ambele private.

private int degree;  
 private ArrayList<MonomialWithIntegerCoefficient> polynom;

Metodele principale din aceasta clasa sunt cele de adunare, scadere, inmultire si derivare.

Metoda *add* este o metoda suprascrisa si realizeaza adunarea unui polinom cu un monom. Aceasta metoda cauta in lista de monoame a polinomului unul care are acelasi grad ca monomul trasmis ca parametru. In cazul in care un astfel de monom exista in lista se va realiza adunarea monoamelor, altfel se va adauga parametrul functiei in lista de monoame, cu metoda add a ArrayList-ului. La final se parcurge de doua ori lista de monoame a polinomului. Prima parcurgere recalculeaza gradul polinomului iar in a doua parcurgere se sterg din lista monoamele care au coeficientul 0.

public void add(MonomialWithIntegerCoefficient a) {  
 int ok=0;  
 for (MonomialWithIntegerCoefficient j : this.polynom)  
 if (a.getDegree() == j.getDegree()) {  
 j.add(a);  
 ok=1;  
 break;  
 }  
 if(ok==0)  
 this.polynom.add(a);  
  
 this.degree=0;  
 for (MonomialWithIntegerCoefficient i: this.polynom) {  
 if(i.getDegree()>this.degree && i.getCoefficient()!=0)  
 this.degree=i.getDegree();  
 }  
  
 ok=1;  
 while(ok==1) {  
 ok=0;  
 for (int i = 0; i < this.polynom.size(); i++) {  
 if (this.polynom.get(i).getCoefficient() == 0) {  
 this.polynom.remove(i);  
 ok=1;  
 }  
 }  
 }  
 }

Metoda *add* mai are o functionalitate, realizata prin suprascriere, care aduna doua polinoame. Astfel se parcurg listele de monoame ale celor doua polinoame iar data se gasesc monoame cu acelasi grad, ele vor fi adunate. Monoamele, din lista polinomului primit ca parametru, care nu se pot aduna, deoarece nu se gaseste nici un monom cu grad similar, vor fi adaugate cu metoda add a ArrayList-ului. La final se parcurge de doua ori lista de monoame a polinomului. Prima parcurgere recalculeaza gradul polinomului iar in a doua parcurgere se sterg din lista monoamele care au coeficientul 0.

public void add(PolynomialWithIntegerCoefficient a) {  
 for (MonomialWithIntegerCoefficient i: a.polynom) {  
 int ok=0;  
 for (MonomialWithIntegerCoefficient j : this.polynom)  
 if (i.getDegree() == j.getDegree()) {  
 j.add(i);  
 ok=1;  
 break;  
 }  
 if(ok==0) {  
 MonomialWithIntegerCoefficient m=new MonomialWithIntegerCoefficient(i.getDegree(), i.getCoefficient());  
 this.polynom.add(m);  
 }  
 }  
  
 this.degree=0;  
 for (MonomialWithIntegerCoefficient i: this.polynom) {  
 if(i.getDegree()>this.degree && i.getCoefficient()!=0)  
 this.degree=i.getDegree();  
 }  
  
 int ok=1;  
 while(ok==1) {  
 ok=0;  
 for (int i = 0; i < this.polynom.size(); i++) {  
 if (this.polynom.get(i).getCoefficient() == 0) {  
 this.polynom.remove(i);  
 ok=1;  
 }  
 }  
 }  
 }

Metoda *subtract* realizeaza scaderea a doua polinoame. Aceasta metoda parcurge listele de monoame ale celor doua polinoame iar data se gasesc monoame cu acelasi grad, ele vor fi scazute. Monoamele, din lista polinomului primit ca parametru, care nu se pot scadea, deoarece nu se gaseste nici un monom cu grad similar, vor fi adaugate cu semn schimbat cu metoda add a ArrayList-ului. La final se parcurge de doua ori lista de monoame a polinomului. Prima parcurgere recalculeaza gradul polinomului iar in a doua parcurgere se sterg din lista monoamele care au coeficientul 0.

public void subtract(PolynomialWithIntegerCoefficient a) {  
 for (MonomialWithIntegerCoefficient i: a.polynom) {  
 int ok=0;  
 for (MonomialWithIntegerCoefficient j : this.polynom)  
 if (i.getDegree() == j.getDegree()) {  
 j.subtract(i);  
 ok=1;  
 break;  
 }  
 if(ok==0) {  
 MonomialWithIntegerCoefficient m=new MonomialWithIntegerCoefficient(i.getDegree(), (-1) \* i.getCoefficient());  
 this.polynom.add(m);  
 }  
 }  
  
 this.degree=0;  
 for (MonomialWithIntegerCoefficient i: this.polynom) {  
 if(i.getDegree()>this.degree && i.getCoefficient()!=0)  
 this.degree=i.getDegree();  
 }  
  
 int ok=1;  
 while(ok==1) {  
 ok=0;  
 for (int i = 0; i < this.polynom.size(); i++) {  
 if (this.polynom.get(i).getCoefficient() == 0) {  
 this.polynom.remove(i);  
 ok=1;  
 }  
 }  
 }  
 }

Metoda *multiply* realizeaza inmultirea a doua polinoame. Aceasta metoda parcurge listele de monoame ale celor doua polinoame si inmulteste fiecare monom din prima lista cu fiecare monom din a doua lista. Rezultatele obtinute vor fi adunate folosind metodele prezentate mai sus pentru a crea rezultatul inmultirii. La final se parcurge de doua ori lista de monoame a polinomului rezultat. Prima parcurgere recalculeaza gradul polinomului iar in a doua parcurgere se sterg din lista monoamele care au coeficientul 0.

public void multiply(PolynomialWithIntegerCoefficient a) {  
 PolynomialWithIntegerCoefficient p = new PolynomialWithIntegerCoefficient();  
 for (MonomialWithIntegerCoefficient i: a.polynom) {  
 for (MonomialWithIntegerCoefficient j : this.polynom) {  
 MonomialWithIntegerCoefficient m = new MonomialWithIntegerCoefficient(i.getDegree(),0);  
 m.add(i);  
 m.multiply(j);  
 PolynomialWithIntegerCoefficient q = new PolynomialWithIntegerCoefficient();  
 q.polynom.add(m);  
 q.degree = m.getDegree();  
 p.add(q);  
 }  
 }  
 this.polynom = p.polynom;  
 this.degree = p.degree;  
  
 this.degree=0;  
 for (MonomialWithIntegerCoefficient i: this.polynom) {  
 if(i.getDegree()>this.degree && i.getCoefficient()!=0)  
 this.degree=i.getDegree();  
 }  
  
 int ok=1;  
 while(ok==1) {  
 ok=0;  
 for (int i = 0; i < this.polynom.size(); i++) {  
 if (this.polynom.get(i).getCoefficient() == 0) {  
 this.polynom.remove(i);  
 ok=1;  
 }  
 }  
 }  
 }

Metoda *derive* realizeaza derivarea polinomului. Deoarece derivarea unui polinom reprezinta derivarea mai multor monoame, aceasta operatie se realizeaza simplu dupa regula de derivare a unui monom. Astfel se parcurge lista de monoame si se deriveaza fiecare in parte. La final se parcurge de doua ori lista de monoame a polinomului rezultat. Prima parcurgere recalculeaza gradul polinomului iar in a doua parcurgere se sterg din lista monoamele care au coeficientul 0.

public void derive() {  
 for (MonomialWithIntegerCoefficient i: this.polynom) {  
 i.setCoefficient(i.getCoefficient()\*i.getDegree());  
 i.setDegree(i.getDegree()-1);  
 }  
  
 this.degree=0;  
 for (MonomialWithIntegerCoefficient i: this.polynom) {  
 if(i.getDegree()>this.degree && i.getCoefficient()!=0)  
 this.degree=i.getDegree();  
 }

int ok=1;  
 while(ok==1) {  
 ok=0;  
 for (int i = 0; i < this.polynom.size(); i++) {  
 if (this.polynom.get(i).getCoefficient() == 0) {  
 this.polynom.remove(i);  
 ok=1;  
 }  
 }

}  
 }

Metoda *createPolynomialFromString* creaza un polinom dintr-un String. Pentru a desparti String-ul primit ca parametru in String-uri de monoame am folosit Regular Expression (Regex) pentru ca mai apoi sa identific coeficientul si exponentul monomului si sa-l creez ca instantiere a clasei monoamelor. Apoi aceste monoame au fost adaugate in lista de monoame a polinomului.

public void createPolynomialFromString(String s) {  
 String monomialFormat = "([+-]?[^-+]+)";  
 Pattern p = Pattern.*compile*(monomialFormat);  
 Matcher m = p.matcher(s);  
 while(m.find()) {  
 String monomial=m.group(1);  
 int poz=0, length=monomial.length(), coefficient=0, degree=0, sign=1, ok;  
 if(monomial.charAt(poz)=='-')  
 sign=-1;  
 if(monomial.charAt(poz)=='-' || monomial.charAt(0)=='+')  
 poz++;  
  
  
 if((monomial.charAt(poz)>='a' && monomial.charAt(poz)<='z') || (monomial.charAt(poz)>='A' && monomial.charAt(poz)<='Z'))  
 coefficient=1;  
 else  
 while(poz<length && monomial.charAt(poz)>='0' && monomial.charAt(poz)<='9'){  
 coefficient=coefficient\*10+(int)monomial.charAt(poz)-(int)'0';  
 poz++;  
 }  
 coefficient\*=sign;  
  
  
 if(poz!=length) {  
 poz++;  
 if(poz!=length) {  
 poz++;  
 while(poz<length && monomial.charAt(poz)>='0' && monomial.charAt(poz)<='9'){  
 degree=degree\*10+(int)monomial.charAt(poz)-(int)'0';  
 poz++;  
 }  
 }  
 else if(monomial.charAt(poz-1)=='x')degree=1;  
 }  
  
 MonomialWithIntegerCoefficient a=new MonomialWithIntegerCoefficient(degree,coefficient);  
 this.add(a);  
 }  
 }  
  
}

Metoda *print* returneaza ca String polinomul pe care se executa. Aceasta metoda foloseste print-ul realizat in clasa monoamelor. Pentru a afisa monoamele polinomului in ordine descrescatoare a gradului am folosit o metoda de sortare a ArrayList-ului care necesita o clasa pentru compararea monoamelor in functie de grad (SortByDegreeInteger). Aceasta clasa este realizata in pachetul Periferic.

public String print() {  
 Collections.*sort*(this.polynom, new SortByDegreeInteger());  
 int c=0;  
 String s=new String();  
 for (MonomialWithIntegerCoefficient i: this.polynom) {  
 s=s+i.print(c);  
 c++;  
 }  
 if(this.polynom.isEmpty())  
 s=s+'0';  
 return s;  
 }

***Clasa MonomialWithRealCoefficient:***

Ca atribute, aceasta clasa contine gradul monomului si coeficientul acestuia, ambele private. Diferenta fata de clasa MonomialsWithIntegerCoefficient consta in faptul ca in aceasta clasa coeficientul este un float.

private int degree;  
 private float coefficient;

Metodele principale din aceasta clasa sunt asemanatoare cu cele din clasa MonomialsWithRealCoefficient din punct de vedere algoritmic: add, subtract, multiply, print, printDegree. In plus s-a implementat si metoda divide care imparte doua monoame si va fi folosita pentru operatia de impartire a polinoamelor. In aceasta metoda, pentru a ajuta la debugging, se va afisa in consola un mesaj de eroare in cazul in care se incearca impartirea unui monom cu un altul de un grad mai mare.

public MonomialWithRealCoefficient divide(MonomialWithRealCoefficient a) {  
 if(this.degree<a.degree)  
 System.*out*.println("Error: You are trying to divide a monomial with another one with a greater degree!");  
 else {  
 MonomialWithRealCoefficient r=new MonomialWithRealCoefficient(this.degree - a.degree, (float)this.coefficient / a.coefficient);  
 return r;  
 }  
 return null;  
}

***Clasa PolynomialsWithRealCoefficient:***

Ca atribute, aceasta clasa contine gradul polinomului si un ArrayList de monoame cu coeficient real, ambele private.

private int degree;  
 private ArrayList<MonomialWithRealCoefficient> polynom;

Metodele principale din aceasta clasa sunt cele de adunare, scadere, inmultire si derivare. Acestea sunt la fel ca cele prezentate in clasa PolynomialWithIntegerCoefficient si vor fi folosite pentru impelmentarea metodei divide.

Metoda *divide* realizeaza impartirea a doua polinoame. Catul impartirii va fi returnat de aceasta metoda in timp ce restul va ramane in polinomul pe care s-a apelat metoda. Impartirea se realizeaza astfel: daca gradul deimpartitului este mai mare sau egal decat cel al impartitorului se vor imparti monoamele dominante iar rezultatul se adauga in polinomul ‘cat’ iar apoi tot acel rezultat se va inmulti cu un duplicat al impartitorului care va fi scazut din deimpartit. Procesul se repeat pana cand deimpartitul (restul) va avea un grad mai mic decat impartitorul.

public PolynomialWithRealCoefficient divide(PolynomialWithRealCoefficient a) {  
 PolynomialWithRealCoefficient q = new PolynomialWithRealCoefficient();  
 PolynomialWithRealCoefficient r = this.duplicate();  
 int k=0;  
 while(r.degree>=a.degree) {  
 Collections.*sort*(r.polynom, new SortByDegreeReal());  
 Collections.*sort*(q.polynom, new SortByDegreeReal());  
 MonomialWithRealCoefficient m=r.polynom.get(0).divide(a.polynom.get(0));  
 q.add(m);  
 PolynomialWithRealCoefficient e=new PolynomialWithRealCoefficient();  
 e.add(m);  
 PolynomialWithRealCoefficient t=a.duplicate();  
 t.multiply(e);  
 r.subtract(t);  
  
 r.degree=0;  
 for (MonomialWithRealCoefficient i: r.polynom) {  
 if(i.getDegree()>r.degree && i.getCoefficient()!=0)  
 r.degree=i.getDegree();  
 }  
 int ok=1;  
  
 while(ok==1) {  
 ok=0;  
 for (int i = 0; i < r.polynom.size(); i++) {  
 if (r.polynom.get(i).getCoefficient() == 0) {  
 r.polynom.remove(i);  
 ok=1;  
 }  
 }  
 }  
 }  
 this.polynom=r.polynom;  
 this.degree=r.degree;  
 return q;  
 }

Metoda *duplicate* instantiaza un obiect nou care va avea aceleasi valori cu polinomul pe care s-a apelat metoda. Aceasta metoda a fost realizata pentru a fi folosita la impartirea polinoamelor.

public PolynomialWithRealCoefficient duplicate() {  
 PolynomialWithRealCoefficient p=new PolynomialWithRealCoefficient();  
 for (MonomialWithRealCoefficient i: this.polynom) {  
 MonomialWithRealCoefficient m=new MonomialWithRealCoefficient(i.getDegree(), i.getCoefficient());  
 p.add(m);  
 }  
 return p;  
 }  
  
 public String print() {  
 Collections.*sort*(this.polynom, new SortByDegreeReal());  
 int c=0;  
 String s=new String();  
 for (MonomialWithRealCoefficient i: this.polynom) {  
 s=s+i.print(c);  
 c++;  
 }  
 return s;  
 }

Metoda *integrate* realizeaza integrarea polinomului. Deoarece integrarea unui polinom reprezinta integrarea mai multor monoame, aceasta operatie se realizeaza simplu dupa regula de integrare a unui monom. Astfel se parcurge lista de monoame si se integreaza fiecare in parte. La final se parcurge o data lista de monoame pentru a recalcula gradul polinomului.

public void integrate() {  
 for (MonomialWithRealCoefficient i: this.polynom) {  
 i.setCoefficient(i.getCoefficient()/(i.getDegree()+1));  
 i.setDegree(i.getDegree()+1);  
 }  
  
 this.degree=0;  
 for (MonomialWithRealCoefficient i: this.polynom) {  
 if(i.getDegree()>this.degree && i.getCoefficient()!=0)  
 this.degree=i.getDegree();  
 }  
 }

***Clasa Controller***

In aceasta clasa am realizat legatura dintre interfata grafica si pachetul DataModels. In ea sunt create sase metode care sunt apelate de butoanele din interfata si care realizeaza operatia dorita (add, subtract, multiply, divide, derive, integrate) folosind implementarile realizate in pachetul DataModels.

public void add() {  
 PolynomialWithIntegerCoefficient p1 = new PolynomialWithIntegerCoefficient();  
 PolynomialWithIntegerCoefficient p2 = new PolynomialWithIntegerCoefficient();  
 p1.createPolynomialFromString(polinom\_1.getText());  
 p2.createPolynomialFromString(polinom\_2.getText());  
 p1.add(p2);  
 polinom\_rez.setText(p1.print());  
}  
  
public void subtract() {  
 PolynomialWithIntegerCoefficient p1 = new PolynomialWithIntegerCoefficient();  
 PolynomialWithIntegerCoefficient p2 = new PolynomialWithIntegerCoefficient();  
 p1.createPolynomialFromString(polinom\_1.getText());  
 p2.createPolynomialFromString(polinom\_2.getText());  
 p1.subtract(p2);  
 polinom\_rez.setText(p1.print());  
}  
  
public void multiply() {  
 PolynomialWithIntegerCoefficient p1 = new PolynomialWithIntegerCoefficient();  
 PolynomialWithIntegerCoefficient p2 = new PolynomialWithIntegerCoefficient();  
 p1.createPolynomialFromString(polinom\_1.getText());  
 p2.createPolynomialFromString(polinom\_2.getText());  
 p1.multiply(p2);  
 polinom\_rez.setText(p1.print());  
}  
  
public void divide() {  
 PolynomialWithIntegerCoefficient p1 = new PolynomialWithIntegerCoefficient();  
 PolynomialWithIntegerCoefficient p2 = new PolynomialWithIntegerCoefficient();  
 p1.createPolynomialFromString(polinom\_1.getText());  
 p2.createPolynomialFromString(polinom\_2.getText());  
 PolynomialWithRealCoefficient r1 = p1.createPolynomialWithRealCoefficient();  
 PolynomialWithRealCoefficient r2 = p2.createPolynomialWithRealCoefficient();  
 PolynomialWithRealCoefficient cat = r1.divide(r2);  
 if(r1.print().length()==0)  
 polinom\_rez.setText(cat.print());  
 else  
 polinom\_rez.setText(cat.print() + " rest " + r1.print());  
}  
  
public void derive() {  
 PolynomialWithIntegerCoefficient p1 = new PolynomialWithIntegerCoefficient();  
 p1.createPolynomialFromString(polinom\_1.getText());  
 p1.derive();  
 polinom\_rez.setText(p1.print());  
}  
  
public void integrate() {  
 PolynomialWithIntegerCoefficient p1 = new PolynomialWithIntegerCoefficient();  
 p1.createPolynomialFromString(polinom\_1.getText());  
 PolynomialWithRealCoefficient r=p1.createPolynomialWithRealCoefficient();  
 r.integrate();  
 polinom\_rez.setText(r.print()+"+c");

}

**5. Rezultate**

Testarile aplicatiei le-am realizat de mana. Am considerat toate felurile de polinoame care pot fi introduse: de la polinomul cu grad 0 (EX: 4) pana la polinoame cu un grad foarte mare (EX: x^103+x^2+1) si am aplicat pe ele toate operatiile create. Desi in aplicatie nu am considerat cazul in care utilizatorul introduce date gresite, pentru date corecte aplicatie ofera un raspuns la fel de correct.

**6. Concluzie**

In timpul implementarii acestui proiect am invatat multe despre ceea ce inseamna sa treci prin toate etapele de dezvoltare a unei aplicatii si sa inveti lucruri noi cand ai nevoie de ele (Regex, JavaFX).

**7. Bibliografie**

* <https://en.wikipedia.org/wiki/Regular_expression>
* <https://en.wikipedia.org/wiki/JavaFX>