  
Facultatea: Facultatea de Automatică si Calculatoare  
Specializarea: Calculatoare și Tehnologia Informației  
Catedra: Calculatoare  
Titlul proiectului: Calcul de polinoame  
Numele studentului: Nyámcz Simon-István-Balázs  
Numărul grupei: 30236  
Data: 22.03.2020.

# Cuprins

[Cuprins 2](#_Toc35896221)

[Cerințe funcționale 3](#_Toc35896222)

[Obiectivul temei 4](#_Toc35896223)

[Analiza problemei 5](#_Toc35896224)

[Proiectare 6](#_Toc35896225)

[ Decizii de proiectare 6](#_Toc35896226)

[ Proiectare clase 6](#_Toc35896227)

[ Interfata utilizator 6](#_Toc35896228)

[Implementare 7](#_Toc35896229)

[Clasa Monom 7](#_Toc35896230)

[Clasa Polinom 8](#_Toc35896231)

[Clasa Calculator 10](#_Toc35896232)

[Clasa GUI 10](#_Toc35896233)

[Clasa Controller 11](#_Toc35896234)

[Clasa Main 11](#_Toc35896235)

[Rezultate 12](#_Toc35896236)

[Concluzii 13](#_Toc35896237)

[Bibliografie 13](#_Toc35896238)

# Cerințe funcționale

* Propunerea, proiectarea și implementarea unui calculator de polinoame, care să permită efectuarea următorelor operații:
  + Citirea polinoamelor de la tastatură sub forma:  
    anX^n + an-1X^n-1 +...+ a0X^0 ex:. 2X^4 - 4X^2 - 3X^1 + 5X^0
  + Adunare
  + Scădere
  + Înmulțire
  + Împărțire
  + Derivare
  + Integrare
* Realizarea interfeței grafice pentru
  + Întroducereea polinoamelor
  + Selectarea operațiilor care trebuie effectuate asupra polinoamelor
  + Afișara rezultatului
  + Java Swing pentru implementarea interfeței grafice de utilizator
* Folosirea unui design OOP (obiject-oriented programming)
  + Încapsulare
  + Definirea claselor Polinom și Monom
* Folosirea MVC (Model View Controller)
* Regex (regular expression) pentru validarea polinoamelor
* Utilizarea Listelor
* Utilizarea foreach in locul for(int i=0…)
* Implementarea claselor cu maximum 300 de linii de cod
* Implementarea metodelor cu maximum 30 de linii de cod
* Utilizarea convențiile de denumire Java
* Folosirea JUNIT pentru testarea aplicației

# Obiectivul temei

Obiecticul principal: Proiectarea și implementarea unui calculator polinomial cu o interfață grafică dedicată prin intermediul căreia utilizatorul poate introduce polinoame, selecta operația care trebuie efectuată (adică adunarea, scăderea, înmulțire, împărțire, derivare, integrare) și afișarea rezultatului.

Obiective secundare:

* Dezvoltarea de use-case -uri și scenarii -detalii în capitolul următor
* Împărțirea pe clase : proiectul este structurat pe cinci clase, împărțite în trei pachete astfel: clasele Monom, Polinom și Calculator sunt incluse în pachetul Model, clasa Controller este inclusă în pachetul Controller, iar clasa GUI este inclusă în pachetul View -detalii în capitolul Proiectare/Implementare
* Implementarea soluției -detalii în capitolul Implementare
* Testare -detalii în capitolul Rezultate

# Analiza problemei

Se cere proiectarea și implementarea unui calculator polinomial cu o interfață grafică dedicată prin intermediul căreia utilizatorul poate introduce polinoame, selecta operația care trebuie efectuată (adică adunarea, scăderea, înmulțire, împărțire, derivare, integrare) și afișarea rezultatului.

Use Case

* user-ul a introdus polinomul / polinoamele pe care vrea să le folosească
* user-ul indică faptul că dorește efectuarea unei anumite operații pe polinomul / polinoamele introduse prin apăsarea butonului corespunzător acesteia
* pulinomul / polinoamele sunt citite și sunt validate de către parser și transpuse în obiecte de clasă Polinom
  + Polinoam valid are forma:  
    anX^n + an-1X^n-1 +...+ a0X^0 ex:. 2X^4-4X^2-3X^1+5X^0
  + Polinomul fiind alcatuită dintr-o listă de monoame, el trebiue introdus in ordinea descrescatoare a gradurilor monoamelor.
  + Polinomul trebuie introdus cu toți coeficienții/exponenții săi nenuli, inclusiv „1”, pentru a asigura funcționarea corectă a calculatorului de polinoame)
* va fi selectată și efectuată operația cerută
  + Adunare ( A + B )
  + Scădere ( A – B )
  + Înmulțire ( A \* B )
  + Împărțire ( A / B )
  + Derivare ( ∂ A , ∂ B )
  + Integrare ( ∫ A , ∫ A )
* se furnizează un rezultat
* polinomul rezultat este afișat în text field
* dacă se dorește efectuarea unei alte operații , use case-ul se reia

# Proiectare

## Decizii de proiectare

Împărțirea proiectului pe pachete am făcut cu ajutorul architectural patternul-lui MVC: modelul ( package model - clasa Monom, clasa Polinom și clasa Calculator ) , view-ul (package view – clasa GUI) și controller-ul ( package controller – clasa Controller).

## Proiectare clase

Pentru realizarea acestui proiect am folosit 6 clase. Clasa Monom si clasa Polinom pentru stocarea datelor (coeficienți și gradurile monoamelor din polinoame), clasa Calculator pentru realizarea operațiilor pe polinoame (mai precis, intre Monoame), clasa GUI pentru interfața grafică de utilizator, clasa Controller pentru legarea interfeței grafice (View-ul) cu Modelul (Polinom-Monom), și clasa Main, care folosesc pentru rularea aplicației.

## Interfata utilizator

Interfața utilizator este una foarte simplă, ușor de utilizat. Este alcătuită dintr-o fereastră în cadrul căreia am pus un panou cu elementele sale ( etichete: ”A:”, ”B:”, ”Result:”, butoane: ( A + B ), ( A – B ), ( A \* B ), ( A / B ), ( ∂ A ), ( ∂ B ), ( ∫ A ), ( ∫ A ), text fields: firstPolnom, secondPolinom, calcSolution), permite o interacțiune usoară a utilizatorului cu programul ce efectuează operații pe polinoame .

# 

# Implementare

Proiectul este structurat in 5 clase, după cum urmează:

* 3 clase aparținând modelului: clasa Monom, clasa Calculator și clasa Polinom
* O clasă aparținând interfeței cu utilizatorul: clasa GUI
* O clasă aparținând controllerului: clasa Controller
* O clasă pentru rularea applicației:clasa Main

## Clasa Monom

package Model ;  
  
public class Monom {  
  
 private int coefficient ;  
  
 private int grade ;  
  
 public Monom ( int coefficient ){ … }  
  
 public int getCoeficient ( ) { … }  
  
 public void setCoeficient ( int coefficient ) { … }  
  
 public int getGrade ( ) { … }  
  
 public void setGrade ( int grade ) { … }  
  
 public String toString ( ) { … }  
}

Clasa Monomdescrie monoamele din cadrul unui polinom ( care sunt de forma coef X ^exp ), având ca atribute coeficientul și exponentul monomului . Coeficientul este definit ca număr integer, ca și gradul, pentru că nu am reușit să implementez operațiile de Integrare și Împărțire. In cazul în care coeficientul e definit ca număr real ea ar putea fi folosit și pentru a da rezultatele corecte în cadrul operației de integrare, unde coeficientul este împărțit la noul exponent. Gradul unui monom este calculat in cadrul clasei Polinom.

Contructorul cu parametri explicit permite construirea unui monom cu coeficientul dat de noi. Am implementat și metode accesoare și mutatoare pentru a putea avea acces la coeficientul și exponentul monomului și a-l putea modifica, altfel fiind imposibil, atributele fiind declarate private .

Am suprascris metoda toString() în această clasă pentru a putea afișa monoamele în cadrul unui polinom.

## Clasa Polinom

package Model ;  
  
import java.util.ArrayList ;  
  
import java.util.regex.\* ;  
  
public class Polinom {  
  
 private ArrayList < Monom > polinom ;  
  
 public Polinom ( ) { … }  
  
 public Polinom ( Monom [ ] monoms ) { … }  
  
 private ArrayList < Monom > ArrayToArrayList ( Monom [ ] monoms ){ … }  
  
 public void regexChecker( String theRegex, String str2Check) throws NumberFormatException { … }  
  
 public Monom [ ] convertStringToMonomArray ( String string ){ … }  
  
 public ArrayList < Integer > convertStringtoArrayList ( String theRegex, String string ){ … }  
  
 public Polinom plus ( Polinom a ) { … }  
 public Polinom minus ( Polinom a ) { … }  
 public Polinom multiply ( Polinom a ) { … }  
 public Polinom derivate ( ) { … }  
  
 public String toString ( ) { … }  
}

Clasa Polinom, bazându-se pe clasa Monom, descrie un polinom, având ca atribut un ArrayList de monoame, în care apar toate monoamele dintr-un polinom.

Am folosit două tipuri de constructoare. Una implicită, și una în care dau ca și parametru un vector de monoame. La constructorul explicit am utilizat tehnica de fabricare (Factory method) folosind metoda ArrayToArrayList. Această metodă primește ca parametru un vector de monoame. Parcurgând vectorul, pentru fiecare monom calculez gradul lui, care corespunde indexul lui din vector, și-l adaug monomul la un ArrayList, pe care îl returnez.

În cadrul clasei Polinom am verificat corectitudiea stringului (polinomului) introdus de către utilizator în metoda regexChecker. Parcurg stringul str2Check. Până când găsesc substringuri care correspunde cu formatul precizat in theRegex, adaug la stringul result substringul gasit. Dupa ce ies din bucla de căutare, verific, dacă srtingul introdus (str2Check) este acelasi cu stringul creat (cu result). In caz contrar arunc o excepție de tipul NumberFormatException. Primul parametru/string (theRegex) la rândul lui primește valoarea: "(\\+|-)?\\d+[Xx]\\^\\d+". Prima data se verifică daca se începe cu „+” sau cu „-”, sau fără ele „?”. După acesta verifică dacă caracterul („\\d”) sau caracterele următoare („\\d+”) sunt digite. Acesta trebuie urmărit cu litera „x” sau „X”, iar dupa acesta neapărat trebuie să vine caracterul „^”. Iar în final, avănd în vedere faptul că polinomul are la exponent/grad un număr natural, urmează verificarea „\\d+”, adică dacă e digit / sunt digite caracterul / caracterele următoare.

Stringul citit am comvertit in vector de monoame (convertStringToMonomArray). Acestă metodă folosește metoda convertStringtoArrayList pentru covertirea stringului in ArrayList de integer. Acesta primește la fel ca și methoda prezentată mai sus (regexChecker), două stinguri ca parametru: theRegex și string. theRegex în cazul acesta arată în felul următor: "(\\+|-)?\\d+", Unde (\\+|-)? verifică daca digitul („\\d+”-1 sau mai multe) are sau nu semn.Folosind algoritm de parcurgere pentru stringul string prezentată anterior, în acest caz se adugă la ArrayList de integer substrigul găsit, transformat în Integer, astfel ArrayList-ul (numbers) este populată cu coeficienții și exponenții polinomului. Metoda convertStringToMonomArray scoate din ArrayListul numbers elementul al doilea (ceea cu indexul 1), care reprezintă exponentul maxim al polinomului (size). Se declară o variabilă locală de tipul Integer[] (grades), de lungimea găsită anterior (size), care stochează coeficientul fiecărui monom, și ele sunt inițializate cu 0. Parcurg ArrayListul numbers, și pentru fiecare al doilea număr, adică pentru fiecare exponent, în vectorul lor de ”frecvență” (in grades), adaug numarul dinaintea ei.  
Ex: pentru polinomul = 2X^2-3X^0, numbers = (2 2 -3 0), size=numbers(1)=2+1, grades=(2 0 -3).  
pentru polnomul = 6x^5-3x^2-4x^1, numbers = (6 5 -3 2 -4 1), size = numbers(1)=5+1, grades = (6 0 0 -3 -4 0). Folosind grades putem sa generăm un vector de monoame.

Metodele plus, minus, multiply și derivate folosesc metodele corespunzătoare din clasa Calculator pentru realizarea operațiilor specificate.

* Metoda ”plus”și metoda “minus” primește ca parametru un polinom „a” (o listă de monoame). Calculează exponentul maxim și pentru polinomul „a” și pentru polinomul „this”. Se ia maximul dintre ele (max). Pornește un ciclu while pana cănd contorul i ajunge la max. Pentru fiecare I , adica pentru fiecare exponent, se verifica dacă există exponentul în ambele polinoame („a” și „this”). În cazul in care nu există, se trimite null, altfel trimite monomul corespuzător exponentului i către metoda addMonoms / subMonoms specificată in clasa Calculator. Monomul returnat de metoda addMonoms / subMonoms este adaugată in vectorul de monoame monoms, care dupa terminarea ciclului while, este trimis ca și paramertu către constructorul explicit a clasei Polinom.
* Metoda „multiply” primește un polinom („a”) ca și parametru. Calculează exponentul final în urma înmulțirii, adică adună exponentele maxime ale polinoaelor „this” și „a” din care scade 1. Populează/inițializează o listă de monoame de lungime max (result), cu coeficienți 0. Parcurge simultan polinoamele „a” și „this”. Pentru monoamele la exponentul i+j se calculează coeficientul, adunând la aceasta produsul coeficientelor ale monamelor la care suma exponeților este egală cu i+j. Rezultatul (result) este transformat in vector de monoame și este trimis ca și paramertu către constructorul explicit a clasei Polinom.
* Metoda „derivate” calculează exponentul final (max) al polinomului „this”, adică scade din exponentul maxim unu. Populează / inițializează o listă de monoame de lungimea calculată (result), cu coeficienți 0. Parcurge polinomul „result”. Pentru fiecare monom din „result”, se calculează coeficientul: se înmulțește coeficientul monomuli din ”this” care are exponentul i+1. cu exponentul monomuli din ”this” corespunzător. Rezultatul (result) este transformat in vector de monoame și este trimis ca și paramertu către constructorul explicit a clasei Polinom.

Am suprascris și aici metoda toString, alegând să afișez polinoamele folosindu-mă de metoda toString deja suprascrisă în clasa Monom . Am avut în vedere anumite situații speciale de monom cum ar fi cel cu coeficient 0, cu coeficientul monomului ( pozitiv respectiv negativ).

## Clasa Calculator

package Model;  
import java.util.Comparator;  
  
public class Calculator {  
  
 public static Monom addMonoms ( Monom a, Monom b ) { … }  
  
 public static Monom subMonoms ( Monom b, Monom a ) { … }  
  
 public static Monom mulMonoms ( Monom a, Monom b ) { … }  
  
 public static Monom derMonoms ( Monom a ) { … }  
}

Clasa Calculator este folosită pentru a ușura efectuarea operațiilor de adunare, scadere, înmulțire și derivare, astfel încăt să nu trebuie să chemăm methoda corespunzătoare dintr-un monom, ci să avem o posibilitate să adunăm două monoame primite ca parametru.

## Clasa GUI

package View;  
import javax.swing.\*;  
import java.awt.event.ActionListener;  
  
public class GUI extends JFrame {  
  
 private JLabel a = new JLabel ( "A:" ) ; { … }  
 private JTextField firstPolinom = new JTextField ( 24 ) ; { … }  
 private JButton plusButton = new JButton ( "A+B" ) ; { … }  
 public GUI ( ) { … }  
  
 public String getFirstPolinom ( ) { … }  
 public String getSecondPolinom ( ) { … }  
 public String getCalcSolution ( ) { … }  
 public void setCalcSolution ( String solution ) { … }  
  
 public void addListener ( ActionListener listenForButtons ) { … }  
 public void displayErrorMessage ( String errorMessage ) { … }  
}

Am realizat o interfața utilizator foarte simplă, ușor de utilizat. Clasa GUI are atributele:

* etichete: ”A:”, ”B:”, ”Result:”
* butoane: ( A + B ), ( A – B ), ( A \* B ), ( A / B ), ( ∂ A ), ( ∂ B ), ( ∫ A ), ( ∫ A )
* text fields: firstPolnom, secondPolinom, calcSolution.

Am implementat metode accesoare și mutatoare pentru atributele firstPolinom, secondPolinom, calcSolution. Pentru fiecare buton am adăugat acelesi ActionListener, astfel încât in clasa Controller să pot să fac diferență între butoane (pentru fiecare buton am adăugat un text).

## Clasa Controller

package Controller;

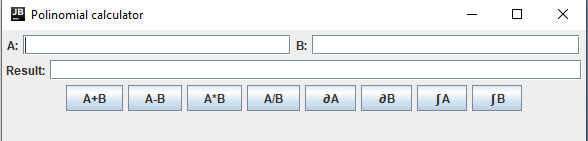
import Model.Polinom;  
import View.GUI;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class Controller {  
 private GUI theView;  
 private Polinom theModel;  
  
 public Controller(GUI theView,Polinom theModel){ }  
  
 class CalculateListener implements ActionListener{  
  
 public void actionPerformed(ActionEvent ex) { … }  
 }  
}

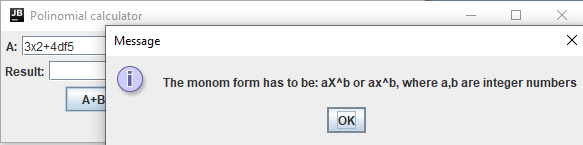
Clasa Controller leagă calsa Polinom și clasa GUI care sunt antributele lui. Constructorul explicit primește ca și parametru o instanță a clasei GUI și una a clasei Polinom. În interiorul clasei Controller am facut încă o clasă, numită CalculateListener, care implementează ActionListener. In aceasta clasa verific corectitudinea stringurilor citite in firstString și secondString (cele din clasa GUI), verific care buton a fost selectată, și pe baza butonului, folosesc metoda corespunzătoare din clasa Polnom.

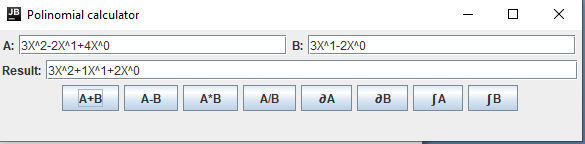
## Clasa Main

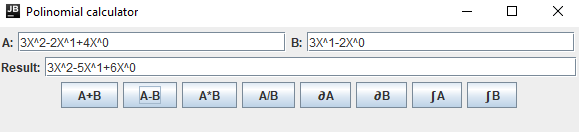
import Controller.Controller;  
import Model.Monom;  
import Model.Polinom;  
import View.GUI;  
  
public class Main {  
 public static void main(String[] args){  
 GUI theView=new GUI();  
 Polinom poli= new Polinom();  
 Controller theController=new Controller(theView,poli);  
 theView.setVisible(true);  
 }  
}

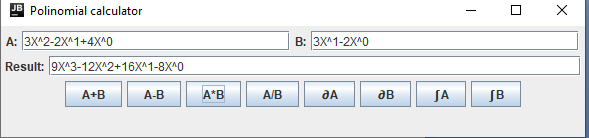
# Rezultate

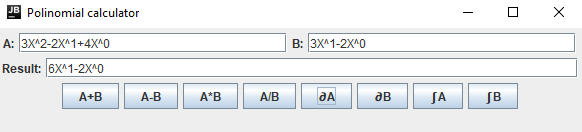


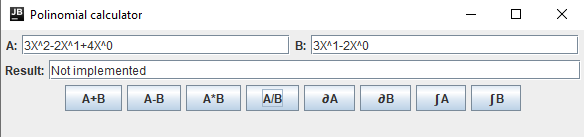












# Concluzii

Pentru a realize acest proiect a trebuit să ințeleg principallele concepte de OOP, precum incapsularea, moștenirea, instanțierea, dr și cele generale, cum ar fi relațtia între clasă și obiect, constructorul, metodele.

Acest calculator încă nu funcționează total correct, deoarece nu am implementat împărțirea și integrarea. Precum nu am folosit nici JUNIT pentru testarea funcționalității. Ulterior se mai pot adauga funcționaitățiile menționate mai sus, și se pot schimba nterfața grafică.

# Bibliografie

www.coned.utcluj.ro/~salomie/PT\_Lic/

http://users.utcluj.ro/~igiosan/teaching\_poo.html

https://github.com/itaisoudry/HW/blob/master/src/il/ac/tau/cs/sw1/ex6/Polynomial.java