|  |
| --- |
| Laboratory Work-Assignment 1 Polynomial Processing |
| Tatar Mara30425March 11, 2016 |

# Abstract

Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

# Introduction

## Task objective

We were asked to build a system able to process polynomial operations such as addition, subtraction, multiplication ,division ,integration and derivation. The system required a Graphical User Interface(GUI).It also has to follow the rules of Object Oriented Programming ,especially the MVC(Model View Control) technique .

## Personal approach

This paper aims to create an intuitive user application,in which the user could perform multiple operations on two polynomials.The result will be displayed on the screen ,using a proper and user friendly interface.

# Problem description

## Problem analysis

The analysis of a problem starts from examining the real model or the model we confront with in the real world and passing the problem through a laborious process of abstractization. Once we identify the subjects of our problem,we can create modules and organize the classes needed.Creating an UML diagram before starting the actual implementation is beneficial to assure a coverage of all the scenarios and a good organization of modules.However, during the implementation of the actual code,

I have encountered multiple issues that aroused from the restriction on introducing only polynomials with integer coefficients.I will come back on this issue later in the paper.

The problem domain is defined by the mathematical definition of a polynomial.A polynomial is an expression that can have constants, variables and exponents, that can be combined using addition, subtraction, multiplication and division, but:  
  
• no division by a variable.  
• a variable's exponents can only be 0,1,2,3,... etc.  
• it can't have an infinite number of terms.

Below is it’s mathematical form:

http://zone.ni.com/images/reference/en-XX/help/371361J-01/eneral_form_of_a_polynomial.xml_d3487e23.png , ,where an≠0, n∈N

In analyzing the problem domain,we notice that the input coefficients of the polynomial and the degrees as integers.The polynomial is obtained from a sum of terms defined by the integer coefficient and the integer degree tuple.In performing certain operations such as addition, subtraction, multiplication and division we will need two polynomials P(x) and Q(x) defined as above. However,the resulting polynomial may not be of the same type,as terms with real coefficient might result from the division operation. In the case of performing operations like integration and derivation on a certain polynomial P(x) ,the derivation operation would always return a polynomial with integer coefficients.It is not the case for the integration operation,that might return a polynomial with real coefficient terms.

## Modeling

The purpose is to implement a system that works with polynomials – from this, we immediately deduce that the main entity that we need to model is the Polynomial.

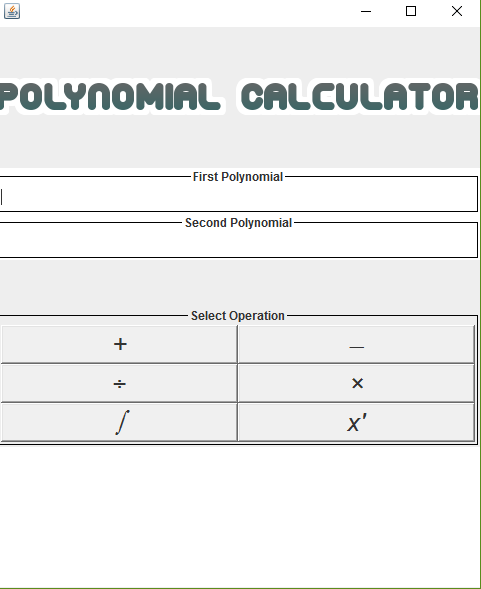
However, the core cell of the program is not the polynomial itself,but the terms that form it.Therfore ,I have declared an abstract class Term ,that contains the coefficient and the degree of each single term of the coefficient.Two more classes were needed to extend the term class,because we would encounter both integer coefficient terms and real coefficient terms.Therefore I have added IntCoeffTerm and RealCoeffTerm,as classes that extend the Term class.

From here I added the Operation class. Even though it was related to the Polynomial class, it was a comprehensive and organized way to work with my polynomials.The class contains multiple static methods that do the operations that were required.

I have left the GUI design in the end and it was then when I realized I might be in trouble as my way of implementing using generics required separate methods to do the conversion from polynomial to string. I have designed the Frame class which deals with the Graphical User Interface and the Converter class which helps me convert from the String input the user introduces to the desired polynomial and the other way around.

## Use cases

The application is as user friendly as possible.The user introduces the polynomials he wants to perform operations on n the 2 boxes presented in the GUI, named First Polynomial and Second Polynomial,respectively.Then there are 6 buttons on the display and he can choose among them which operation to perform:addition,subtraction,division,multiplication,integration or derivation.Once he has selected the desired operation,the answer will appear under the operations box

.

## Scenarios

I will present the possible scenarios for two given polynomials.

For example, the user introduces in the two boxes the following polynomials: P(X) :x^3-12x^2-42 and Q(X): :x-3.

If he presses the first button (+) the application performs the addition and below the buttons the result is printed: x^3-12x^2+x-45.For subtraction he presses (-) and the result is x^3-12x^2-x-39.For division:the quotient Q: x^2-9x-27 and the remainder is R:-123.For integration: 0.25x^4-4x^3-42x and for derivation 3x^2-24x.Regardless of the second polynomial introduced,the calculator will display the integration and derivation for the first polynomial.

# Projection

## *UML diagrams*

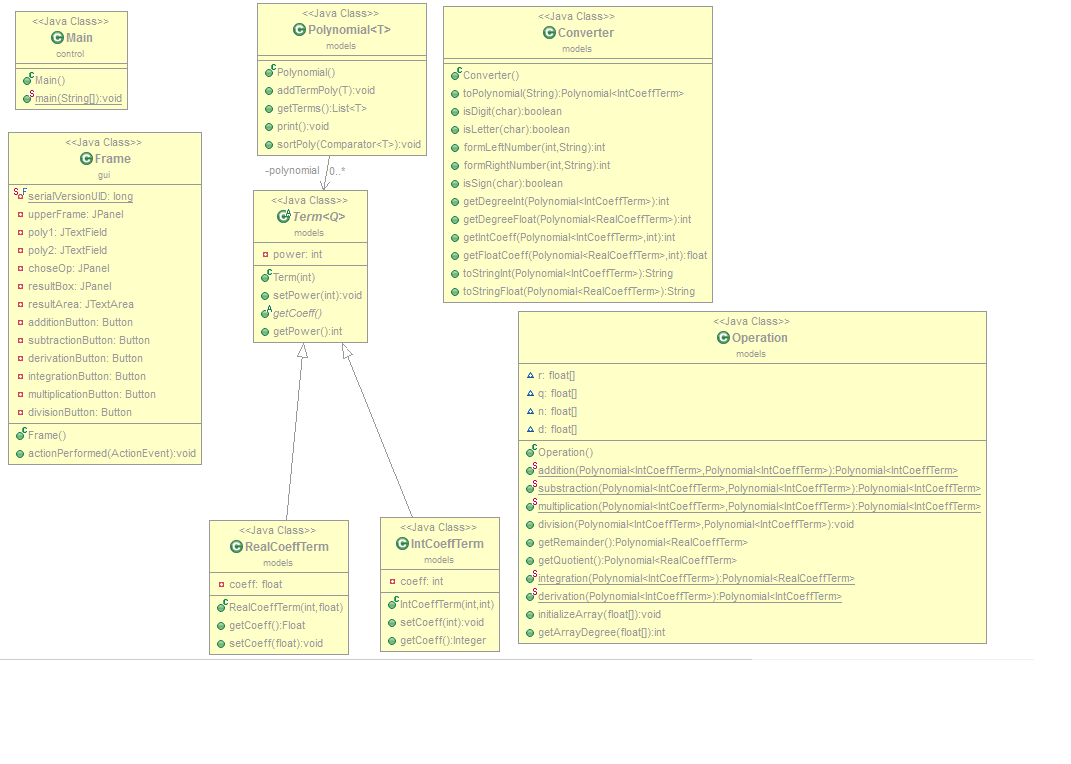
### *Use Case Diagram*

Polynomial **calculator**

**User**

The use case diagram presents the actor, which in our case is the user that uses our application.He performs an operation on the polynomials.

UML Diagram



The project consists of 8 classes,the Main class,the Term abstract class,the IntCoeffTerm class,the RealCoeffTerm class,the Operation class,the Converter class and the Frame class.The relationships between classes will be discussed later in the documentation.

# Packages

## GUI(graphic User Interface)

This class corresponds to the view in the MVC design pattern.It contains the next instance variables:

* + private static final long *serialVersionUID* = 1L;-this is generated automatically
  + private JPanel upperFrame;--the upper part of the Frame,divided in three parts ,corresponding to the title,the box where the user introduces the first polynomial and the box where he/she introduces the second polynomial;
  + private JTextField poly1; - we need to retrieve the first polynomial from the string introduced in this text box;
  + private JTextField poly2; - we need to retrieve the second polynomial from the string introduced in this text box;
  + private JPanel choseOp; -the middle part of the Frame,contains the 6 buttons with the operations
  + private JPanel resultBox; -the lower part of the Frame,contains the JTextArea where the result is displayed
  + private JTextArea resultArea; - needs to be declared globally because of the Action Listener,displays the result
  + private Button additionButton, subtractionButton, derivationButton, integrationButton, multiplicationButton, divisionButton; - these buttons are used for performing the operations: addition, subtraction, multiplication, division, differentiation, integration.

The constructor of this class divides the Frame into equal parts.For the upper part,it sets an icon image and it initializes the two text fields later used to retrieve the polynomials.I have used the title border to indicate the user where should he/she place the first and second polynomial,respectivele ,for the scenarios where the order of insertion matters( example:for division).

The constructor also places the buttons in a panel using Unicode codes for each operation(except for derivation).I have added ActionListeners to each button,to assure the responses to the clicks of the user.

It also sets the resultArea at the bottom,and at last it sets the visibility of the frame,the size and the close operation.

## B)Models

# 1 Term Class

The Term<Q> class is an abstract class that uses generics to implement both terms with integer and float coefficients. This term is defined as stated below in the Problem Analysis section by its coefficient and its degree. However ,in the Term class I have defined only one instance variable:

**Private int power**

Aside from the constructor,which sets the power based on a int variable it receives(which is basically the degree of the term) ,it also has a getter method,a setter for the power and an abstract method which gets the coefficient of generic type Q.

**Public void setPower(int power);**

**Public abstract Q getCoeff ();**

**Public int getPower();**

# 2 IntCoeffTerm Class

IntCoeffterm extends Term<Integer>. Its constructor calls the parent constructor to set the power and also sets the integer coefficient.The method getCoeff is overridden and returns a type Integer.the methods in this class are

**Public void setCoeff();**

**Public Integer getCoeff();**

# 2 RealCoeffTerm Class

RealCoeffterm extends Term<Integer>. Its constructor calls the parent constructor to set the power and also sets the real coefficient.The method getCoeff is overridden and returns a type Float.the methods in this class are

**Public void setCoeff();**

**Public Float getCoeff();]**

# Polynomial Class

This class was introduced for obtaining a polynomial after the introduction of several term.It is a generic class,containing elements of type T (which extends Term).It has one instance variable:

**Private List<T> polynomial**

The constructor initializes the ArrayList of T objects.The motivation for using ArrayLists to represent the polynomial was that they are dynamic and save memory not just by occupying a dynamic space,but also by not introducing terms of power 0 and coefficient 0,as it was the case of arrays.

It contains the following methods:

**Public void addtermPoly(T term); -adds a term to the arraylist**

**Public void print()-prints the polynomial in the console,redundant once the gui was designed**

**Public void sort poly();-places the terms in ascending order of their degrees**

# Operation Class

This class is the place where we place the operation methods.It has no explicit constructor,because we don’t have to instantiate objects belonging to this class.Even though it might be related to the Polynomial ,placing the operations methods in this class looks more organized and it was more intuitive for me to do it in this way.In this class I have encountered multiple issues related to the usage of ArrayLists instead of simple arrays.for example,both the addition methods and the substraction methods take place in O(n^2) time,because I have to iterate through both the arraylists in order to add the terms that have the same degree.Also,once I have done that,there is still the possibility that there are terms of a certain degree in one polynomial that do not appear in the second polynomial,and we also have to add them (or subtract them) to the resulting polynomial.This also involves a “for in for” technique,because we need to compare the first polynomial and second polynomial to the polynomial we have previously obtained. Addition and subtraction both return polynomials of integer coeffiecient terms.

**Public static Polynomial<IntCoeffTerm> addition();**

**Public static Polynomial<IntCoeffTerm> subtraction ();**

The multiplication method also involved some unexpected issues.I tried to continue using the arraylists ,but the resulting array stored terms of same degree as different terms. The goal was to update the coefficient of one term once the iterator found one of the same degree.However,after I have converted them to array that stored at index I the coefficient of the term with degree I,the task was a piece of cake,at the expense of poor memory manage.

**Public static Polynomial<IntCoeffTerm> multiplication()**;

The division method was a real challenge as well.to simplify my task,I have recurred to the same method of using arrays instead of arraylists.I have used the long division algorithm,which required an extra method getArrayDegree.

A pseudocode of the method is described below:

degree(**P**):

**return** the index of the last non-zero element of **P**;

if all elements are 0, return -∞

polynomial\_long\_division(**N**, **D**) *returns* (**q**, **r**):

*//* ***N****,* ***D****,* ***q****,* ***r*** *are vectors*

**if** degree(**D**) < 0 **then** *error*

**if** degree(**N**) ≥ degree(**D**) **then**

**q** ← **0**

**while** degree(**N**) ≥ degree(**D**)

**d** ← **D** *shifted right* *by* (degree(**N**) - degree(**D**))

**q**(degree(**N**) - degree(**D**)) ← **N**(degree(**N**)) / **d**(degree(**d**))

*// by construction, degree(****d****) = degree(****N****) of course*

**d** ← **d** \* **q**(degree(**N**) - degree(**D**))

**N** ← **N** - **d**

**endwhile**

**r** ← **N**

**else**

**q** ← **0**

**r** ← **N**

**endif**

The method is a static void because we want to retrieve both the quotient and the remainder.Two extra methods were needed for that getQuotient and getRemainder.

**Public static void division();**

**Public static Polynomial<RealCoeffterm> getQuotient();**

**Public static Polynomial<RealCoeffterm> getRemainder();**

The remaining integration and derivation methods used arraylists and were relatively easy.The former returned a polynomial of real coefficient terms,and the latter a polynomial of integer coefficient terms.They operate on a single polynomial.

**Public static Polynomial<RealCoeffterm> integration();**

**Public static Polynomial<RealCoeffterm> derivation();**

# Converter Class

This class is core to link the Graphical User Interface to the functional part of the application.It is used to convert the String that the user introduced in the frame to an object of type Polynomial,and from an object of type Polynomial to a String displayed in the result box in the GUI.

Its main methods are toPolynomial ,toStringInt and toStringFloat.Different methods were needed to turn the polynomial in a String because of the variant nature of the polynomial(int and real coefficients).The methods were rather demanding to implement,because they involve lots of cases to cover.For example,in the case of translating a String in a polynomial,we need to identify terms in the form **coeff\* x^power** ,terms in the form **x^power** ,terms in the form **coeff** and to transform them accordingly.

Helping methods were required ,such as:

**Public Boolean isDigit(char a);**

**Public Boolean isLetter(char a);**

**Public Boolean isSign(char a);**

**Public int formLeftNumber(int i,String input);**

**Public int formRightNumber(int i,String input);**

For the public **Polynomial<IntCoeffterm> toPolynomial**(String input) method and

**Public getDegreeInt(Polynomial<IntCoeffterm> polynomial)**

**Public getDegreeFloat(Polynomial<RealCoeffterm> polynomial**)

**public** **int** **getIntCoeff**(**Polynomial** <IntCoeffTerm> polynomial, **int** pow)

**public** **float** **getFloatCoeff**(**Polynomial** <RealCoeffTerm> polynomial, **int** pow)

for the **public String toStringInt(Polynomial <IntCoeffTerm> p)** and the **public String toStringFloat(Polynomial <RealCoeffTerm> p)** methods.

# C)Comparator

This package contains two classes PowerComparatorFloat and PowerComparatorInt which extend Comparator and override the method compareTo.They are used in sorting the polynomials with the sortPoly method in the Operation class.

# D)Control

Contains the Main class

public static void main(String[] args) {

new Frame();

}

Which creates an object of type Frame .

# Algorithms

# Mathematical alghorithms

Addition and subtraction of two polynomials are performed by adding or subtracting corresponding [coefficients](https://en.wikipedia.org/wiki/Coefficients). If

f(x) = \sum_{i=0}^n a_ix^i; g(x) = \sum_{i=0}^m b_ix^i

then addition is defined as

f(x)+g(x)= \sum_{i=0}^m (a_i+b_i)x^i where m > n

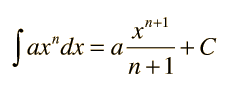
Multiplication is performed much the same way as addition and subtraction, but instead by multiplying the corresponding coefficients. If f(x) = \sum_{i=0}^n a_ix^i; g(x) = \sum_{i=0}^m b_ix^ithen multiplication is defined as f(x)\times g(x)=\sum_{i=0}^{n+m} c_ix^iwhere c_k=a_0b_k+a_1b_{k-1}+\cdots+a_{k-1}b_1+a_kb_0. Note that we treat a_ias zero for i>nand that the degree of the product is equal to the [sum](https://en.wikipedia.org/wiki/Summation) of the degrees of the two polynomials.

Division follows the algorithm I have explained in the previous section ,the Long Division algorithm.

The [integral](http://hyperphysics.phy-astr.gsu.edu/hbase/integ.html#c2) of any polynomial is the sum of the integrals of its terms. A general term of a polynomial can be written

http://hyperphysics.phy-astr.gsu.edu/hbase/imgmth/ipol1.gif

and the [indefinite integral](http://hyperphysics.phy-astr.gsu.edu/hbase/intdef.html#c2) of that term is



where a and C are constants.

The methods that implement the mathematical algorithms were previously discussed in the classe Operation.

# Implementation and testing

In what the implementation is concerned this project was developed in Eclipse and it was only tested in this environment. However the program should maintain its portability.

The algorithms used in this project were not of great difficulty,however linking the project all together was a bit tedious when trying to make it as efficient as possible and when trying to follow the rules of object oriented programming .The usage of Collections proved to be both a benefit and a limitation at the same time and has generated some unexpected issues among the implementation.

Testing implies checking for any errors in the program or limitations it may impose.i have tried to cover all the possible scenarios ,however there are always some small bugs that the developer fails to see at a first glance.Hopefully,I have succeeded in covering the possible scenarios.The errors that could arise are related to the GUI,not the implemented algorithms. The GUI demans for a little expertise coming from the user,as it is not proofed when it comes to the user introducing strings that do not respect the polynomial format.This will remain as a future development.

# What have I learned

After finishing this project, I realized that planning ahead of coding is an extremely good idea.Due to my lack of enough experience coding by the OOP rules,I have failed in designing and planning the project before the actual implementation. When designing the GUI, drawing the window and all the inner components on paper made it all significantly easier. Also, another conclusion would be that an unspecified assignment gives a lot of freedom but is also a lot harder to implement .Rousseau said that “Too much freedom is a dangerous thing”.I don’t think he targeted programmers,but I applies perfectly.

Also,time management was a big challenge,as there were hinderances at every step and solving them before the due date proved to be harder than expected.As future developments,I intend to plan my project in such manner that I get to cover all the scenarios and have a clean ,well organized and easy to maintain code.

# Results and conclusions

The application is user-friendly and succeeds in performing basic polynomial operations such as addition,subtraction,multiplication ,division,differentiation and integration. Further development is possible ,as the Graphical User Interface can be designed to be more appealing to the user and also more informative.For example,a menu could be added containing the a Help option for those who want to look up something related to themathematical form of the polynomial.Also,the calculator could store a number of previous results for the case in which he/ she would want to work later on those polynomials. Also,we can extend the functionality of the polynomial processor by adding more possibilities to it,such as computing the roots of the polynomial or displaying its graph.Translation,rotation and other operations related to the graph would be also practical in some cases. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . .

. . . . . . . . . . . . . . . .. . . . . . . .. . . . . . .. . .. . . . . . . . .. . . .