Technical University of Cluj-Napoca

Computer Science and Automation Faculty

Computer Science Department

2nd semester 2015-2016

Programming Techniques

Homework 1

Pascaru Cosmina – Roxana

Group: 30425

CONTENTS

1. Assignment Objectives

(EN) Lab – Homework 1

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

The purpose of the project is to design and implement an application that can be used to compute several operations on polynomials:

- Addition
- Subtraction
- Multiplication
- Derivation
- Integration
- evaluation in a point

The operations can be performed on one or two polynomials.

The application provides a graphical user interface through which the user can add the desired polynomials, select the operation they want to be performed and also visualize the result.

2. Problem Analysis. Modeling

The first thing to begin with in solving the problem of polynomial implementation is to choose the way to implement them. One possibility was to create a vector of coefficients and have the index of the numbers in the array as the power. Another possibility was to create the polynomials as arrays of monoms. This was the method used in this project.

Each monom would therefore have a coefficient and a power. The values for these attributes of the monom are introduced by the user of the application.

A polynomial is a separate class which contains an array list of monoms. As each monom is created for a particular polynomial, it is added to the list of monoms contained by that specific polynomial.

The operations are done on polynomials, more specifically on the array list of monoms the polynomial has. The methods of an operation get one or two polynomials as parameters and return a Polynomial.

The graphical user interface of the application makes it possible for the user to introduce polynomials and to select the application that they want to be performed on the polynomials. The polynomials are introduced through text fields. Pressing a button saves the introduced text in a variable. This variable is then divided into tokens corresponding to the coefficients of the polynomial and the powers.

Selecting the other buttons corresponding to specific operations will prompt the result of that operation on the screen through the graphical user interface.

3. Usage scenarios

The application is meant to be used for doing operations on polynomials. These operations include addition, subtraction, multiplication, derivation, integration and evaluation in a point.

The first thing a user has to do in order for the application to work properly is to introduce the order of the polynomial. Then they should introduce the polynomial in the format “c1X^p1 +c2X^p2 +…”. If the coefficients are negative integers they can be simply added with a minus (“-“) sign in front, instead of a plus (“+”). The input would look something like this:

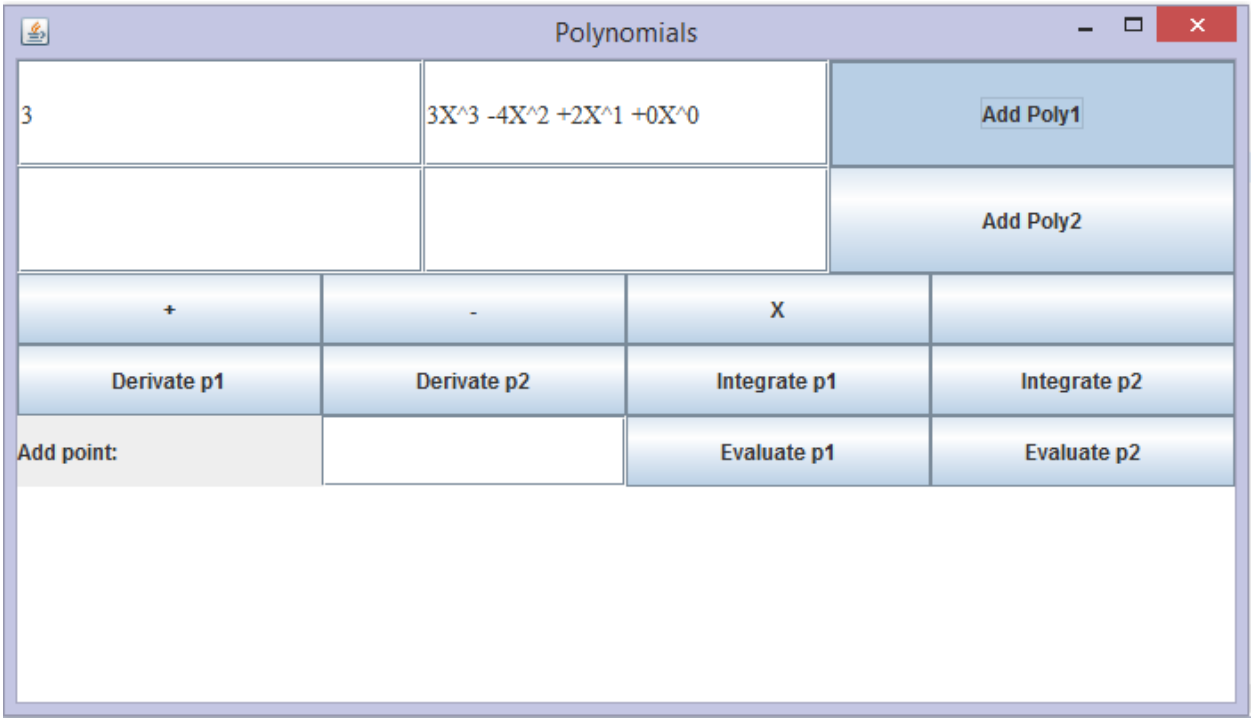
- “2X^2 -7X^1 -6X^0”

for a second degree polynomial. Of course, if the coefficient of the first monom is a negative integer, it can be added with a minus in front (for example “-1X^1 +5X^0”).

If the user wants to introduce a third degree polynomial that only has the coefficient of the X to the third power and the free term different from zero (for example “4X^3 +1”), they would have to enter “4X^3 +0X^2 +0X^1+1X^0”. So even if a monom is not actually present in the polynomial, the application requires tor the user to enter those monoms as well.

A polynomial is saved after the button “Add poly” was pressed. If the user doesn’t press this button, the polynomial cannot be used to perform any operation.

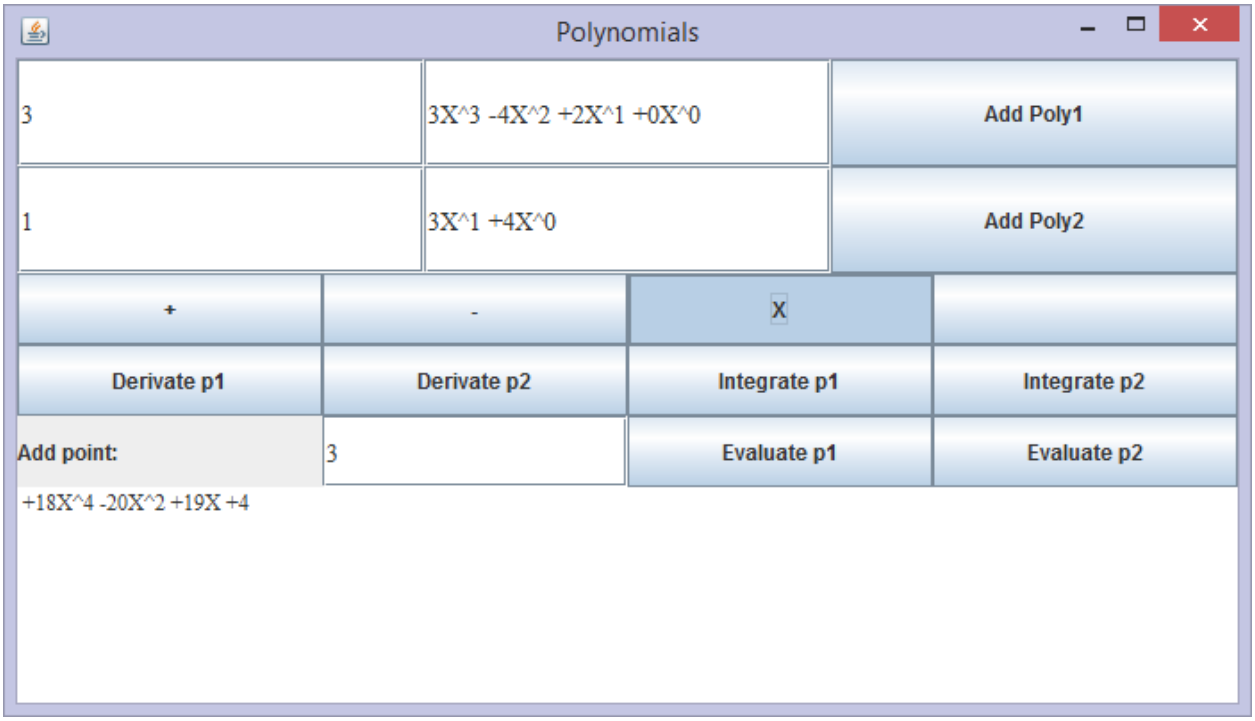
The user can enter polynomials of any degree, and the two polynomials can have different degrees.



In order to evaluate a polynomial at a certain point, the user has to enter an integer (the point) which will replace the X in the coefficient. The input from the reader is saved when the user presses the “Evaluate poly” button. The result will shortly be prompted on the screen.

The graphical user interface of the application displays the results of the operations in a special text area, that is not editable, meaning that the user cannot change the value that is written in this text area.

The result of the operations is displayed in a regular format, meaning that monoms with a zero coefficient will not be displayed, ones with the power zero will only have their coefficient showing, if the coefficient is one, it will not be displayed, and same goes for the power (only ‘X’ will be visible), except for the evaluation in a point. In this case the result is an integer (the value of the polynomial).



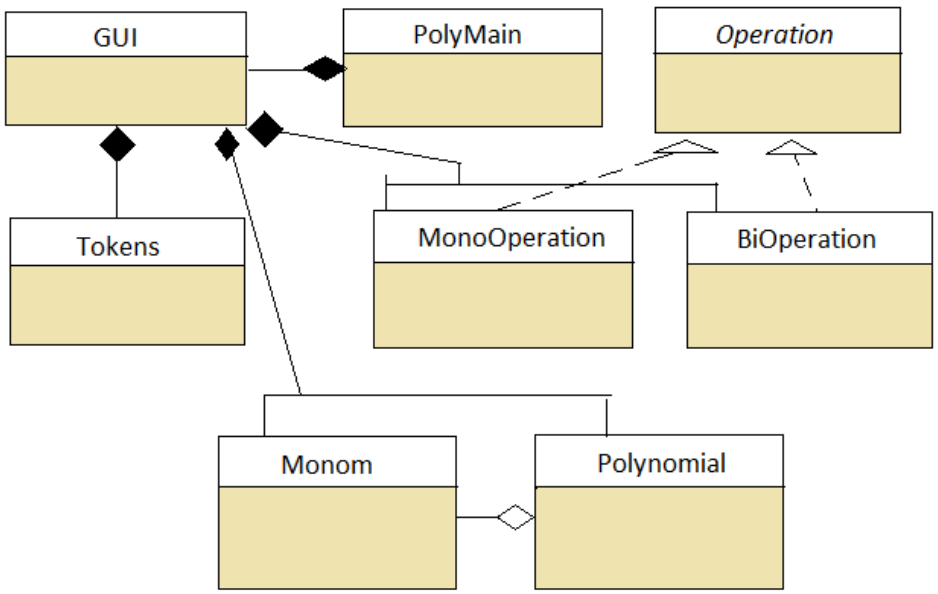
4. Design

The project is divided into seven classes and 3 packages. As discussed before, the polynomials are represented using an arraylist of monoms. In the package `polynomials`, there are the classes defining the polynomials (`Monom` and `Polynomial`).

The operations are defined in the package `operations`. The two classes `MonoOperation` and `BiOperation` implement the `Operation` interface and describe operations that are performed on one, respectively, two polynomials.

The `main` package consists of the `PolyMain` class which has the main method, the `GUI` class which is the one creating the graphical user interface through which the user "communicates" with the application and the `Tokens` class which helps break down the input from the user into monoms.

4.1 UML and Use Case Diagrams



All classes in the `polynomials` project are interconnected. The `PolyMain` class which contains the main method has a reference object of the `GUI` class. This way, when the main function is called, it creates a new `GUI` object `gui` which will automatically call the parameter less constructor of the `GUI` class.

The GUI class uses `Polynomial, Monom` and `Operation` type objects. It implements the `ActionListener` interface, overwriting its method `actionPerformed`.

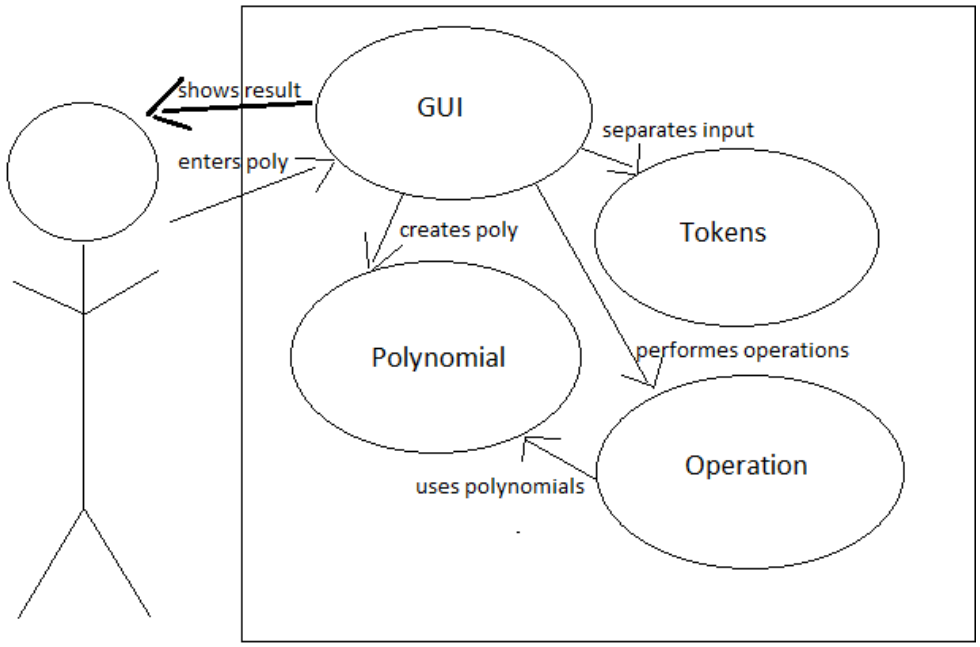
The `Tokens` class operates on the string introduced by the user through the graphical user interface and divides it into coefficients and powers.

The operation classes `MonoOperation` and `BiOperation` implement the `Operation` interface which contains all the methods that an operation can perform, let them be on one or two polynomials. Then, the classes `MonoOperation` and `BiOperation` overwrite only the methods they actually need. So, the `BiOperation` class will only overwrite the add, subtract and multiply methods, leaving the other ones default.

The Use Case diagram captures the dynamic aspect of the system.[2]

The actor is the user of the application. He/She introduces a polynomial and the application has to provide a result for his/hers request.

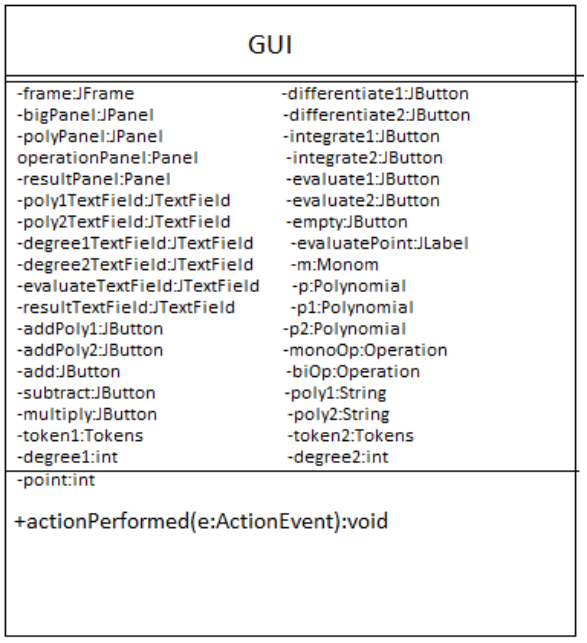
The classes communicate and use other classes in order to satisfy the requirements of the user.



4.2 Classes

4.2.1 GUI class

The GUI class implements the `ActionListener` interface, so it overwrites the method `actionPerformed`(). Its purpose is to create the graphical user interface through which the user of the application is able to enter polynomials, select operations and view the result.



The class contains instances of all the other classes in the project, as well as swing and `awt components` such as `JFrame, JPanel, JButton, JTextField, Layout, Color, Font` and `Dimension.`

Attributes of `GUI` class

- `Frame`
  - `private JFrame frame = new JFrame("Polynomials");`
  - creates the base container for all the other components
  - is has a predefined size and position; it will always appear in the middle of the screen

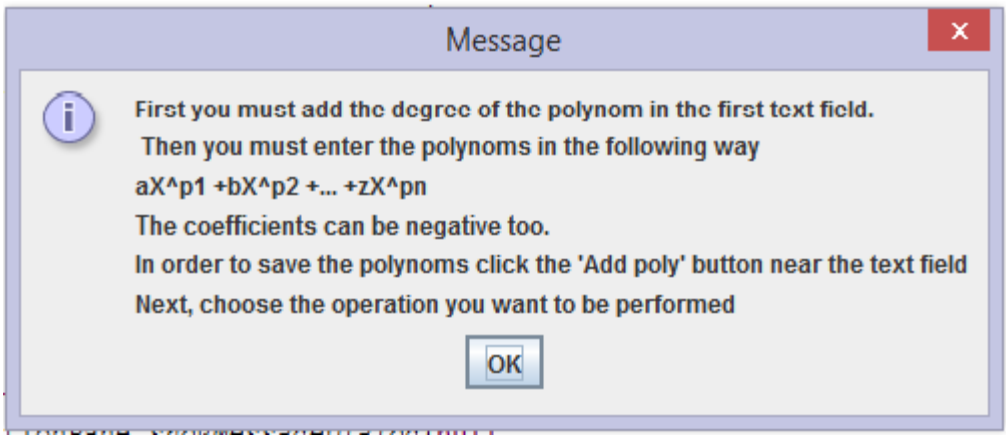
- `Big Panel`
  - `private JPanel bigPanel = new JPanel();`
  - It has a grid layout with 3 lines and one column
  - It contains three other panels (polyPanel, operationPanel and resultPanel)
- `Polynomial Panel`
  - `private JPanel polyPanel = new JPanel();`
  - It has a grid layout with 2 lines and 3 columns
  - It contains four text fields and two buttons (two text fields for entering the degree of the two polynomials, and two for entering the polynomials and finally, two button for saving the polynomials; these buttons must be pressed in order for the application to be able to use the polynomials)

- `Operation Panel`
  - `private JPanel operationPanel = new JPanel();`
  - It has a grid layout with 3 lines and 4 columns
  - It contains the buttons used for selecting the operation to be performed (both `monoOperations` and `biOperations`): add, subtract, multiply, differentiate, integrate and evaluate polynomial at a given point
  - It also contains a text field where the point is entered as an integer
- `Result Panel`
  - `private JPanel resultPanel = new JPanel();`
  - It has a grid layout with one line and one column
  - It contains the result text field, where the user can see the result of the operation

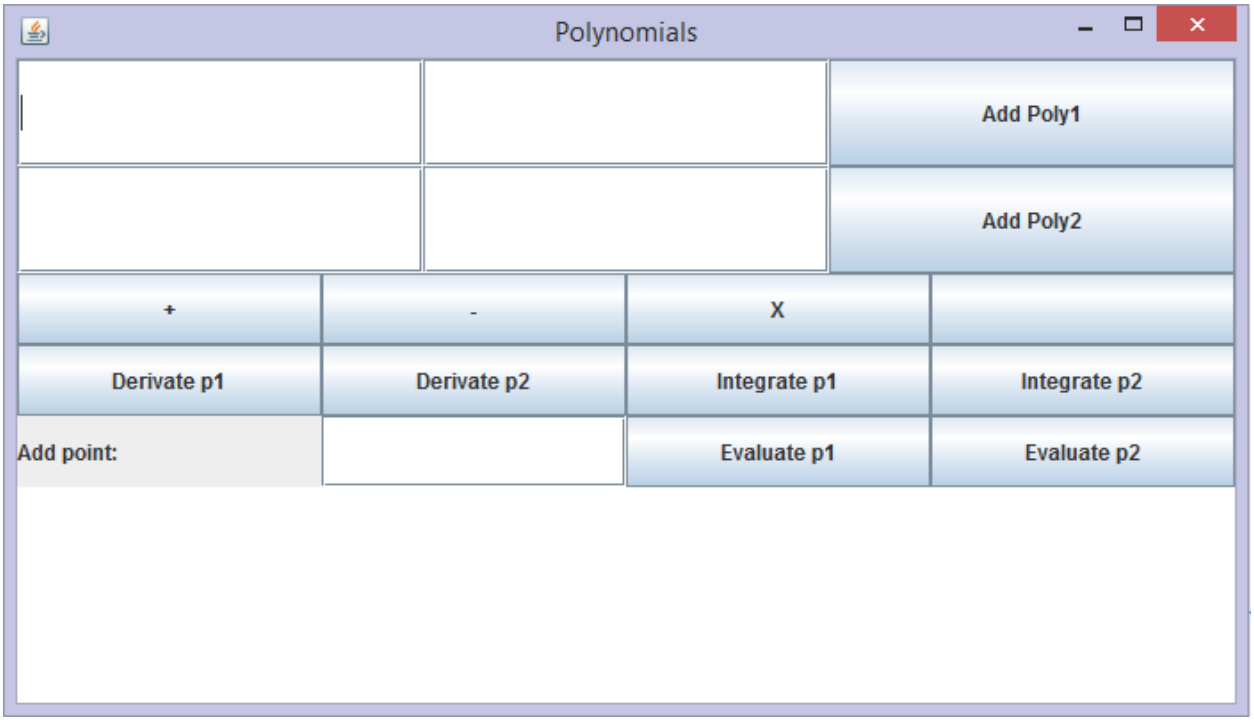
All panels are added to the `bigPanel`, and the `bigPanel` is added to the `frame` at the end of the constructor.

Other attributes include `the` textFields `(poly1TextField, poly2TextField, degree1TextField, degree2TextField, evaluateTextField)`, the result textArea (`resultTextArea`), JButtons `(addPoly1, addPoly2, add, subtract, multiply, differentiate1, integrate1, etc.)`, a label (`evaluatePoint`), strings to hold the entered polynomials before tokenizing them `(poly1, poly2)`, Tokens `(token1, token2)`, Polynomials `(p, p1, p1)` and Operation `(monoOp, biOp)`.

The constructor of the `GUI` class first calls the `showInstructions()` method which at the execution of the program prompts a message dialog containing usage instructions.



After the “OK” button is clicked, the graphical user interface of the application appears on the screen.

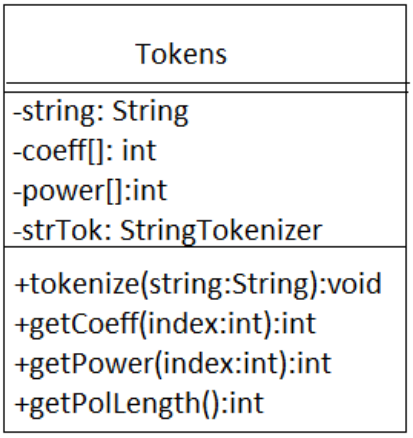


Methods of the `GUI` class

- `showInstructions()`
  - creates a `JOptionPane.messageDialog` which displays some usage instructions on the screen
  - it is meant to guide the user to correctly use the application
  - it is called by the constructor of the GUI class
- `actionPerformed(ActionEvent e)`
  - it is an overwritten method inherited from the interface `ActionListener` which the class implements
  - it adds behavior to the different components of the `GUI` class
  - the selected component is verified in an if-else clause `if(e.getActionCommand().equals(“addPoly1”)){}`

4.2.2 The Tokens class

The `Tokens` class is used to divide the strings of the polynomials given by the user into arrays of coefficients and powers. Then the monoms corresponding to the given polynomial can be created.



The attributes of the Tokens class

- `string : String`
  - passed as a parameter to the constructor of the class
  - it represents the string read from the poly1TextField or poly2TextField
  - also passed as a parameter to the tokenize() method where it is separated into tokens, based on the characters "X^ "
- `strTok: String Tokenizer`
  - the result of the tokenize process
  - contains only the integers from the original string
- `coeff[],power[]:int`
  - arrays of integers representing the coefficients and the powers of the polynomial
  - filled based on an order condition (even elements of the strTok are coefficients and odd ones are powers)

Methods of the Tokens class

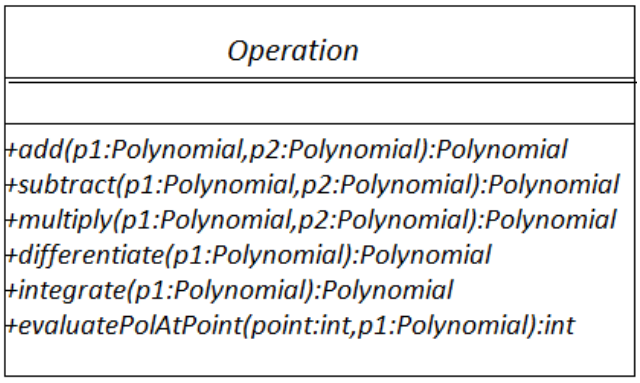
- `tokenize(string:String)`
  - divides the initial string into tokens based on the characters "X^ "
  - separates the resulting tokens into coefficients and powers
  - uses a contor i to check whether the token represents a coefficient (even number) or a power (odd number)

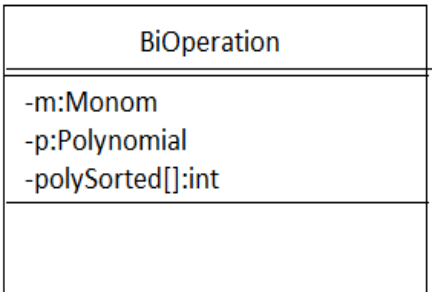
```
int i=0;
while(strTok.hasMoreTokens()){
     if(i%2 ==0){
          coeff[k] =
Integer.parseInt(strTok.nextToken());
          k++;

     }else{

          power[l] =
Integer.parseInt(strTok.nextToken());
          l++;

     }

     i++;

}
```

- `getCoeff(index:int)`
  - returns an integer representing the element at index index in the coefficients array
- `getPower(index:int)`
  - returns an integer representing the element at index index in the powers array
- `getPolLength()`
  - returns the length of any of the array (coeff or power) which also represents the length of the polynomial, or its degree

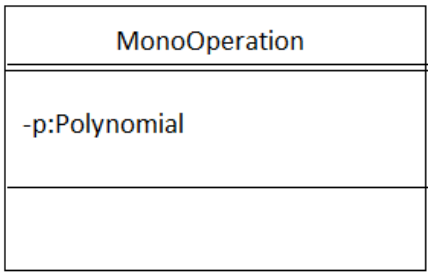
4.2.3 The Operation interface. The MonoOperation and BiOperation classes

The Operation interface contains the abstract methods add, subtract, multiply, differentiate, integrate and evaluate, which represent both operations on one polynomial as well as operations on two polynomials. The methods declared in the interface are all abstract, which means they will have to be overwritten by all the classes that implement the Operation interface.





`Monoperation` and `BiOperation` classes implement the `Operation` interface. The `MonoOperation` class will overwrite only the methods which represent operations on one polynomial from the interface, while the `BiOperation` class will implement the ones representing operation on two polynomials. The other methods will remain default.



Attributes of the MonoOperation class

- `p:Polynomial`
  - is instantiated in the constructor of the class
  - it represents the new polynomial created after the operations are performed

Methods of the MonoOperation class

- `differentiate(p1:Polynomial)`
  - gets a Polynomial as a parameter
  - computes the polynomial resulted after the differentiation of the given polynomial
  - if the power of the Monom from the original Polynomial is different from zero, then it creates a new Monom for each Monom from the original Polynomial by setting its coefficient as the coefficient times the power of the original Polynomial, and the power as the power minus one

```
for(Monom m:p1.getArray()){
     m = new Monom(m.getCoeff() * m.getPower(),
m.getPower()-1);
}
```

  - then it adds the new Monom to the Polynomial p
  - if the power of the Monom from the original polynomial is equal to zero, then it created a new Monom with coefficient and power equal to zero
- `integrate(p1:Polynomial)`
  - based on the same principle as the differentiation, but the coefficient is left the same; only the power is incremented by one
  - the actual computation of the coefficient is done in the actionPerformed() method from the GUI class, where the coefficient is

transformed into a float as is computed as the result of the division of the coeff / power

- `evaluatePolyAtPoint(point:int, p1:Polynomial)`
  - returns an integer
  - computes the result of replacing X with the point in the Polynomial
  - the Math.pow() method is used to raise point to the power of the Monom
  - the result is added to the integer result which will be returned by the method

Attributes of the BiOperation class

- `p:Polynomial`
  - used to store the new created Polynomial resulted from doing the operations
  - is returned by all the methods implemented in the BiOperation class
- `m:Monom`
  - used for creating new Monoms after the Polynomial has been sorted according to powers
- `polySorted[]:int`
  - is an array used to order the Polynomial
  - the index represents the power and the value of the array at that index represents the coefficient

Methods of the BiOperation class

- `add(p1:Polynomial, p2:Polynomial)`
  - adds two polynomials
  - first checks which Polynomial has a bigger degree
  - then for each Monom from the Polynomial p1, it adds the coefficient at index equal to the power in the polySorted array
  - then for each Monom from the Polynomial p2, if the power of p2 is equal to the power of p1, the coefficient of p2 is added to the polySorted array, at the index equal to that power

```
if (p1.getDegree() > p2.getDegree()) {
     for (Monom m1 : p1.getArray()) {
          polySorted[m1.getPower()] = m1.getCoeff();
          }
          for (Monom m1 : p1.getArray()) {
               for (Monom m2 : p2.getArray()) {
                    if (m1.getPower() == m2.getPower()) {
                         polySorted[m1.getPower()] +=
               m2.getCoeff();
     }}}}}
```

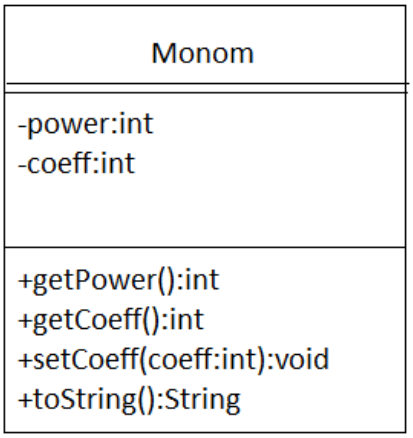
- the process is similar if the degree of the second Polynomial is greater
- if their degrees of the Polynomials are equal, then a simple addition is done whenever the powers are equal

- `subtract(p1:Polynomial,p2:Polynomial)`
  - computes the difference of the two Polynomials
  - algorithm similar to the one from addition
- `multiply(p1:Polynomial,p2:Polynomial)`
  - computes the product of two Polynomials
  - simply creates a new Monom with the coefficients from the two Polynomials multiplied, and their powers added and then sorts the Monoms by using the polySorted array
  -

4.2.4 The Monom class

The `Monom` class creates an abstract representation of a monom. It is used in the Polynomial class to create an ArrayList of Monoms.

The constructor of the Monom class has two parameters, coeff and power, both integers, which are used to create a Monom of the form “coeffX^power”.



Attributes of the Monom class

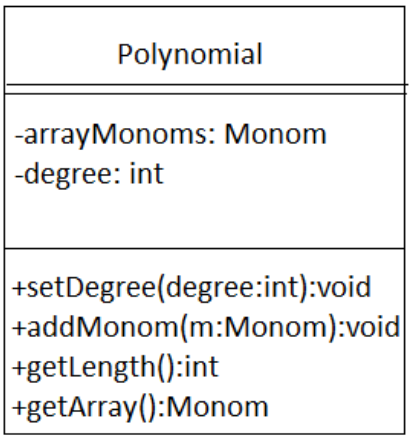
- `coeff:int`
  - is an integer representing the coefficient of a monom
  - is set in the constructor of the class which receives an integer with the same name as a parameter
    this.coeff = coeff;
- `power:int`
  - is an integer representing the power of a monom
  - it is set in the constructor of the class which receives an integer with the same nameas a parameter
    this.power = power

Methods of the Monom class

- `getPower()`
  - Returns an integer representing the power of the Monom
- `getCoeff()`
  - returns an integer representing the coefficient of the Monom
- `setCoeff(coeff:int)`
  - sets the coefficient of the Monom equal to the parameter coeff
- `toString()`
  - Returns a String representing the form of the Monom
  - The cases are solved in an if – else statement
  - Some of the cases are :
    - Return an empty string if the coefficient and the power of the Monom are zero or if the coefficient if equal to zero
    - Return the coefficient with a "+" in front if the power of the Monom is zero and the coefficient is positive, or with a "-" if the coefficient is negative
    -

4.2.5 The Polynomial class

The `Polynomial` class creates an abstract representation of a polynomial. It contains an arrayList of Monoms which forms the polynomial.



Attributes of the Polynomial class

- `arrayMonoms:ArrayList<Monom>`
  - represents an arrayList of Monoms which forms the Polynomial
  - it is instantiated in the constructor of the Polynomial class
- `degree: int`
  - represents the degree of the polynomial
  - it can be set with the methos setDegree()

Methods of the Polynomial class

- `setDegree(degree:int)`
  - sets the degree of the Polynomials equal to the parameter degree that is received by the method
- `addMonom(m:Monom)`
  - adds a Monom to the arrayMonoms
- `getLength()`
  - return the length of the Polynomial
- `getArray()`
  - return the arrayMonoms ArrayList
- `getDegree()`
  - returns the degree of the Polynomial

4.3Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier.[1]
The Polynomials project consists of three packages: main, operations and polynomials.

`Main` Package contains:

- the class `PolyMain` which has the main `main()` method
- the class `GUI` which creates the graphical user interface and implements the `ActionListener` interface
- the class `Tokens` which separates the string introduced by the user into coefficients and powers

`Operations` Package contains:

- the interface `Operation`
- the class `MonoOperation` which has methods that perfom operations on one Polynomial
- the class `BiOperaton` which has methods that perform opretation on two Polynomials
- both classes implement the Operation interface

`Polynomials` Package contains:

- the class `Monoms`
- the class `Polynomial`

Imported Packages:

- java.awt.Color;
- java.awt.Dimension;
- java.awt.Font;
- java.awt.GridLayout;
- java.awt.event.ActionEvent;
- java.awt.event.ActionListener;
- javax.swing.JButton;
- javax.swing.JFrame;
- javax.swing.JLabel;
- javax.swing.JOptionPane;
- javax.swing.JPanel;
- javax.swing.JTextArea;
- javax.swing.JTextField;

5. Results

The implementation of the Polynomials project has as results the possibility to compute operations on any degree polynomials. It can be used for fast computations of the addition, subtraction and multiplication of two polynomials, as well as for fast differentiation and integration of a polynomial.

6. Conclusions. Further implementations

Working at the Polynomials project has been a complex job, due to the numerous computations and the displaying of the polynomials. There were many cases to take care of in both these situations.
The reading of the polynomials was also challenging. However, the implementation with separating the tokens based on the parity of their order was a relatively simple solution.
When it came to ordering the polynomial resulted after the multiplication of the two polynomials entered by the user, using an array with the index representing the power and the content at that index representing the coefficient, seemed to be the best way to solve the problem.
Some further implementation of the application would include:

- being able to enter the polynomial in a closer way to the how they are usually written on paper
- adding new operations that can be performed on polynomials, such as : division, finding the roots of the polynomial or drawing its graph

7.Bibliography

- [1]. http://www.tutorialspoint.com/java/java_packages.htm
- [2]. http://www.tutorialspoint.com/uml/uml_use_case_diagram.htm
- http://stackoverflow.com
- https://docs.oracle.com/javase/7/docs/api/overview-summary.html