Polynomial Operations

Student: Bonta Vlad-Valentin

Group:30424

Contents

1.Objective of the assignment

The objective of this assignment was finding an optimal and clear solution to design a system for Polynomial operations processing (addition, subtraction, multiplication, division, integration, differentiation). The Polynomials have one variable and integer/real coefficients.

2.Analysis of the problem, modeling, scenarios, use cases

2.1. Analysis of the problem

A polynomial is an expression consisting of variables and coefficients. The general form of a polynomial is: $a_n \times \mathrm{X}^m + a_{n-1} \times \mathrm{X}^{m-1} + \cdots + a_0 \times \mathrm{X}^0$, where a_n, a_{n-1},.... a_0 are the coefficients and X is the variable.[1] The number m gives the degree of the polynomial. A polynomial can have one or more terms like $a_n \times \mathrm{X}^m$ consisting of a coefficient(a_n), a variable(X) and the degree of the term (m). In this project, polynomials can have only one variable, so we do not retain the variable. At the first sight, it was a good idea to keep a list with every degree from the highest order to 0 but that would be a bad idea regarding the memory because, for example if the user input is x^{5000}+x^1 the list will have a lot of space filled with terms that have the coefficient and the degree equal with 0. Taking the memory problem into consideration and the general form, I decided to represent the polynomial as a whole just with the given terms, instead of requesting every monome separately so it relates more closely to the real word representation. Every term from polynomial have real coefficient in order to be able to give a better result or integration and a positive integer degree.

2.2. Scenarios

The desired scenario is when the user inputs the data correctly, the data introduced is processed and shown to the user for a better visualization, chooses an operation and the result is computed and the result is returned to the user. But this scenario may not happen all the time. For example, the user can introduce the polynomial in other format than the one specified and the application should be able to handle these situations without any problem.

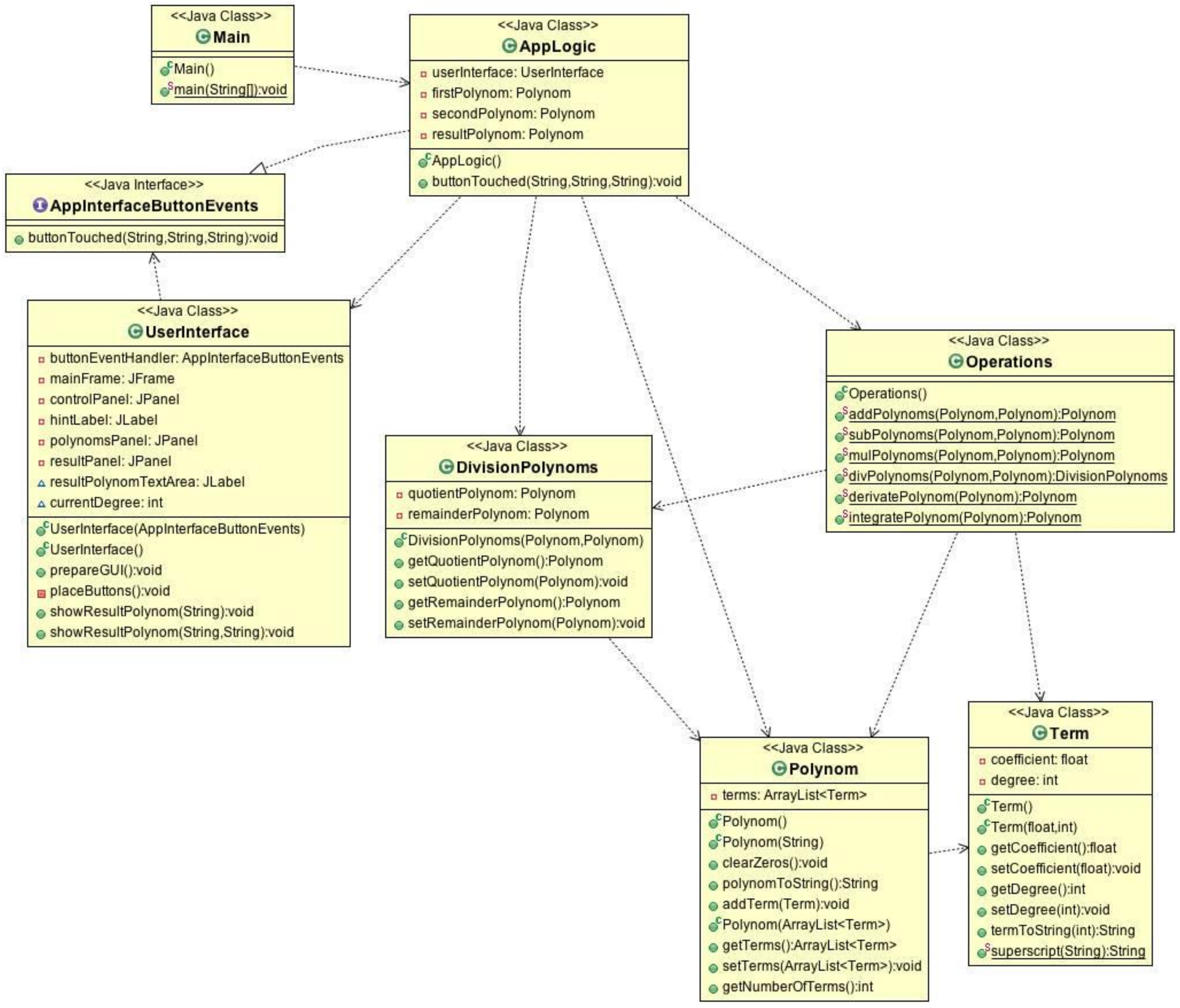
2.3. Use cases

A use case defines the interactions between external actors and the system under consideration to accomplish a goal. Actors must be able to make decisions.[2] According to this definition, we can identify the main actor as the user of application.

Given that the user is the main actor, we can highlight some of the ways he can interact with the application: the user can compute operations on polynomials after he give as an input 2 polynomials. As a main user you can add, subtract, multiply, divide, derivate or integrate the desired polynomials.

3.Design

3.1. UML Class Diagram



<<Java Class>>
Main
Main()
main(String[]):void
<<Java Class>>
AppLogic
userInterface: UserInterface
firstPolynom: Polynom
secondPolynom: Polynom
resultPolynom: Polynom
AppLogic()
buttonTouched(String,String,String):void
<<Java Interface>>
AppInterfaceButtonEvents
buttonTouched(String,String,String):void
<<Java Class>>
UserInterface
buttonEventHandler: AppInterfaceButtonEvents
mainFrame: JFrame
controlPanel: JPanel
hintLabel: JLabel
polynomsPanel: JPanel
resultPanel: JPanel
resultPolynomTextArea: JLabel
currentDegree: int
UserInterface(AppInterfaceButtonEvents)
UserInterface()
prepareGUI():void
placeButtons():void
showResultPolynom(String):void
showResultPolynom(String,String):void
<<Java Class>>
DivisionPolynoms
quotientPolynom: Polynom
remainderPolynom: Polynom
DivisionPolynoms(Polynom,Polynom)
getQuotientPolynom():Polynom
setQuotientPolynom(Polynom):void
getRemainderPolynom():Polynom
setRemainderPolynom(Polynom):void
<<Java Class>>
Operations
Operations()
addPolynoms(Polynom,Polynom):Polynom
subPolynoms(Polynom,Polynom):Polynom
mulPolynoms(Polynom,Polynom):Polynom
divPolynoms(Polynom,Polynom):DivisionPolynoms
derivatePolynom(Polynom):Polynom
integratePolynom(Polynom):Polynom
<<Java Class>>
Polynom
terms: ArrayList<Term>
Polynom()
Polynom(String)
clearZeros():void
polynomToString():String
addTerm(Term):void
Polynom(ArrayList<Term>)
getTerms():ArrayList<Term>
setTerms(ArrayList<Term>):void
getNumberOfTerms():int
<<Java Class>>
Term
coefficient: float
degree: int
Term()
Term(float,int)
getCoefficient():float
setCoefficient(float):void
getDegree():int
setDegree(int):void
termToString(int):String
superscript(String):String

3.2. Data Structures

Polynomial: The Polynom class has as attributes the list of terms. These list represents the coefficients and degrees of the polynomial that the user inserted. As I mentioned previously, these are the only attributes we need to express a term because the variable is always the same.

Other data structures used are the term representation, and the DivisionPolynom class which handles the result of a division and will be discussed later.

3.3. Algorithms

In this section, the algorithms applied for the operations will be discussed in more detail, in order to give a better understanding of the implementation.

Addition

In adding polynomials, the addition is made on two terms if they have the same degree. The result will be a term with the same degree and the coefficient is the sum of the two terms.

For example: $3x + 2x$ is $9x$ and $9x^5 + 2x^3$ is simply $9x^5 + 2x^3$ because the two polynomials do not have any term with the same degree.

This operation is performed by the method public static `Polynom addPolynoms(Polynom firstPolynom, Polynom secondPolynom) in the Operations class.`

The two polynomials to be added are passes as parameters of type Polynomial, and method return also an object of Polynom type which is the sum of the given polynomials.

Subtraction

Subtraction of polynomials is similar with the addition, but instead of adding the polynomials, we subtract them.

This operation is performed by the method `public static Polynom subPolynoms(Polynom firstPolynom, Polynom secondPolynom)from Operations class.`

Multiplication

Multiplying polynomials is done by multiplying each term of the first polynomials with all the terms from the second polynomials and the resulting polynomial is added to the final result.

The multiplication is performed by the method `public static Polynom mulPolynoms(Polynom firstPolynom, Polynom secondPolynom) from Operations class;`

This method requires two parameters of type Polynom and the returning object is of type Polynom representing the multiplication of the given polynomials.

Division

The division of two polynomials is a bit more complicated than the other operations. Division is the reverse of multiplication.

The result of a division consists of the quotient and the remainder polynomials so we need another class to handle the result, so I have implemented the DivisionPolynoms class which has as attributes a quotient and a remainder of type Polynom.

The algorithm of division is made while the second polynomial degree is higher or equal with the first polynomial degree. While this condition is applying, we find the term which multiplied with the first term from the first polynomial gives the first term of the second polynomial. Using this term we will create the quotient of the division. After we find this term, we multiply the first polynomial with it and then substract it from the second polynomial and the result will be assigned to the second polynomial. After this operation, we go back to the first step and repeat the whole process. When the condition is not satisfied anymore, the quotient of the division is created and the remainder equals the second polynomial. [4]

The method that handles the division requires two parameters of type Polynom and the returning object is a DivisionPolynoms type. The implementation is made by the method `public static DivisionPolynoms divPolynoms(Polynom firstPolynom, Polynom secondPolynom)` from `Operation class.`

Derivation

The derivation of a polynomial is made by calculating the derivative of every term with the general form:

$$\frac{d}{dx}x^n = nx^{n-1}$$

From the mathematical definition, we observe that every term will have the coefficient multiplied with the order and the order decreases by one.

The derivation method requires only one parameter of type Polynom and the returning object is also a Polynom object. It is implemented by the method `public static Polynom derivatePolynom(Polynom firstPolynom)` from `Operations class.`

```java
public static Polynom derivatePolynom(Polynom firstPolynom) {
    Polynom resultPolynom = new Polynom();
    int firstPolynomNrOfTerms = firstPolynom.getNumberOfTerms();

    int firstPolynomIndex = 0;

    Term firstPolynomTerm = new Term(0, 0);

    for (firstPolynomIndex = 0; firstPolynomIndex < firstPolynomNrOfTerms;firstPolynomIndex++) {
        firstPolynomTerm = firstPolynom.getTerms().get(firstPolynomIndex);
        firstPolynomTerm.setCoefficient(firstPolynomTerm.getCoefficient()*firstPolynomTerm.getDegree());
        firstPolynomTerm.setDegree(firstPolynomTerm.getDegree()-1);

        resultPolynom.addTerm(firstPolynomTerm);
    }
    return resultPolynom;
}
```

Integration

The integration of a polynomial is made by calculating the integration of every term with the general form:

$$\int ax^n\,dx = a\frac{x^{n+1}}{n+1} + C$$

From the mathematical definition, we observe that every term will have the order added with 1 and the coefficient divided with the order+1.

The integration method requires only one parameter of type Polynom and the returning object is also a Polynom object. It is implemented by the method `public static Polynom integratePolynom(Polynom firstPolynom) from Operations class:`

```java
public static Polynom integratePolynom(Polynom firstPolynom) {
        Polynom resultPolynom = new Polynom();
        int firstPolynomNrOfTerms = firstPolynom.getNumberOfTerms();

        int firstPolynomIndex = 0;

        Term firstPolynomTerm = new Term(0, 0);

        for (firstPolynomIndex = 0; firstPolynomIndex < firstPolynomNrOfTerms;firstPolynomIndex++) {
            firstPolynomTerm = firstPolynom.getTerms().get(firstPolynomIndex);
            firstPolynomTerm.setDegree(firstPolynomTerm.getDegree()+1);
            firstPolynomTerm.setCoefficient(firstPolynomTerm.getCoefficient()/firstPolynomTerm.getDegree());

            resultPolynom.addTerm(firstPolynomTerm);
        }

        return resultPolynom;
    }
```

3.4. Class Design

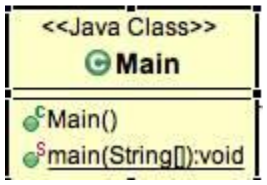
Classes used in project are split into classes regarding the purpose of each class.

(default package)	GUI	Logic	Model	Utils
Main	AppInterfaceButtonEvents	AppLogic	DivisionPolynoms	Operations
	UserInterface		Polynom	
			Term	

Using Maven it structures the project such as you see a clear differentiation between implementation classes and test classes. The resources are also well organized, such that implementation and tests have different folders for resources.

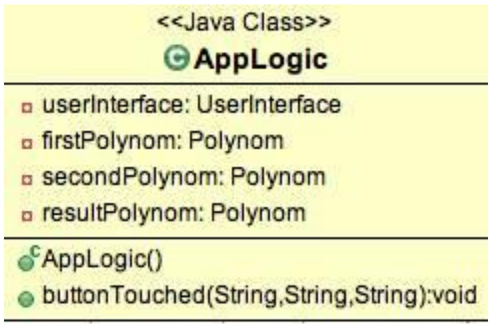
Main Class

The class is used just to instantiate the AppLogic class.



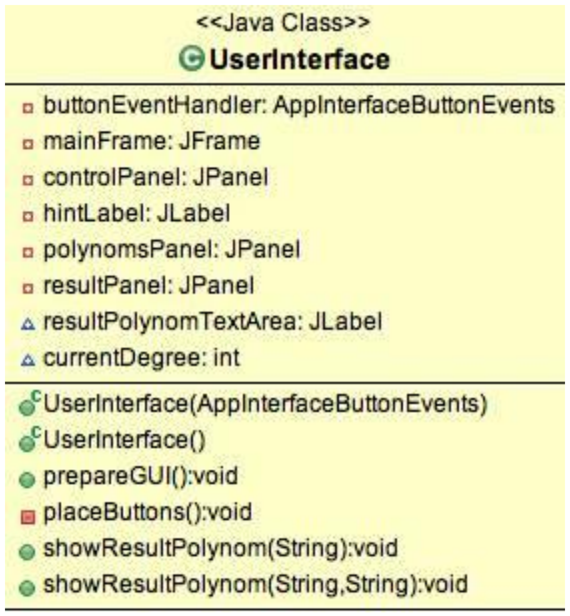
AppLogic Class

AppLogic class is used as the controller of the application. It manages the interface and the polynomials operations. It implements the AppInteracceButtonEvents interface and implements the method buttonTouched(String operationString, String firstPolynomString, String secondPolynomString) which is called by the UserInterface when the user presses a button. The arguments of this method are the name of the pressed button and the polynomials in String format.



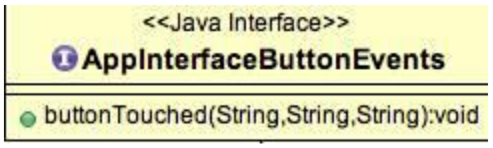
UserInterface Class

UserInterface class manages the user interface of the application. It consists of the operations buttons and text fields for getting the user input and a text field for displaying the result polynomial.



<u>AppInteracceButtonEvents Interface</u>

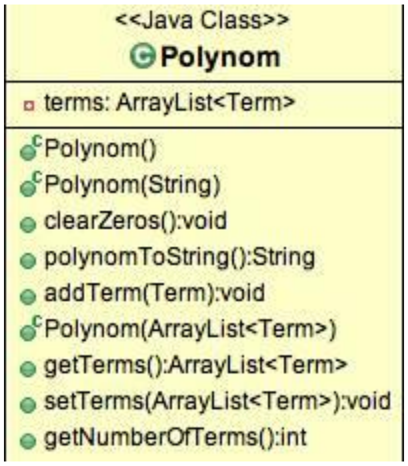
AppInteracceButtonEvents interface has just one method that is implemented by the AppLogic class. This interface works as a delegate between the UserInterface class and AppLogic class. For every button we have an actionListener that we use to know when the user pressed a button and which button.



<u>Polynom Class</u>

Polynom class is used, as stated before, to model the real life representation of a polynomial as a whole. It consists of a list with terms that is private to have control over it. This class has a constructor which receives as a parameter a String representing the polynomial and it parse the string into a Polynom instance. The parsing is made after a regex expression: **"(-?\\b\\d+)[xX]\\^(-?\\d+\\b)"**[3] that searches for the pair "x^" or X^" and what is on the left side is the coefficient and on the right side is the degree. Using this regex, is easily to separate the terms from the provided string, the only inconvenience is that the string should be in descending order of degrees and should have all the terms as: ax^n in order to have a valid polynomial.

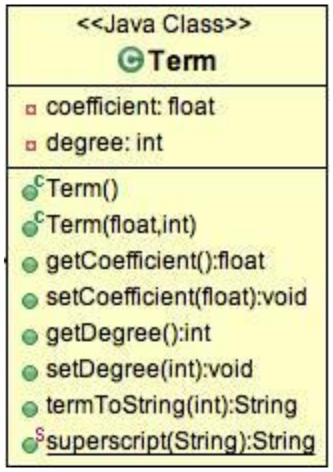
A method which should be mentioned is clearZeros() method which remove from the list the terms that are zero. The method polynomToString() return a String with the lists of terms in printable format. This method is used when we want to print something to user on interface.



Term Class

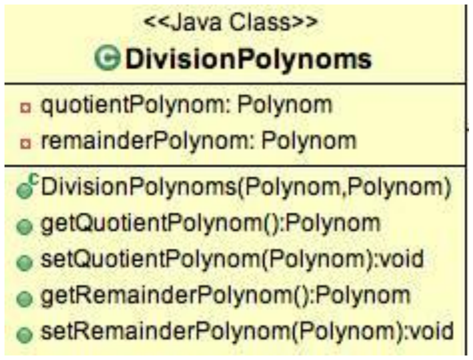
The Term class contains only the coefficient and the degree of the term. A method which should be mentioned is termToString(int index) which handles how the term will be printed. Another useful method is formatFloat(float d) which returns a nice format for float, for example for the number 233.0 it return 233 and for 233.333 it return 233.33.

```
public static String floatFormat(float d){
        if(d == (long) d)
            return String.format("%d",(long)d);
        else
            return String.format("%s",d);
    }
```



DivisionPolynoms Class

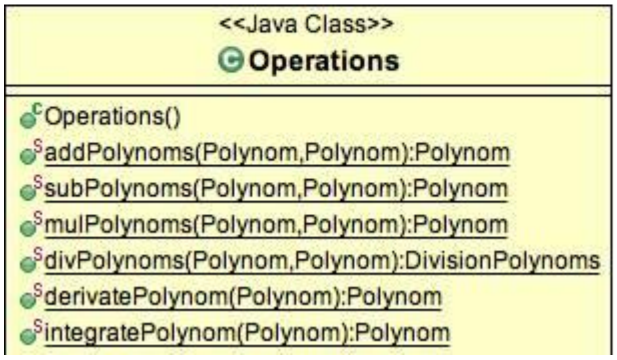
DivisionPolynoms class is used, as mentioned before, for the division operation. It contains the quotient and remainder polynomials.



<<Java Class>>
DivisionPolynoms
quotientPolynom: Polynom
remainderPolynom: Polynom
DivisionPolynoms(Polynom,Polynom)
getQuotientPolynom():Polynom
setQuotientPolynom(Polynom):void
getRemainderPolynom():Polynom
setRemainderPolynom(Polynom):void

Operations Class

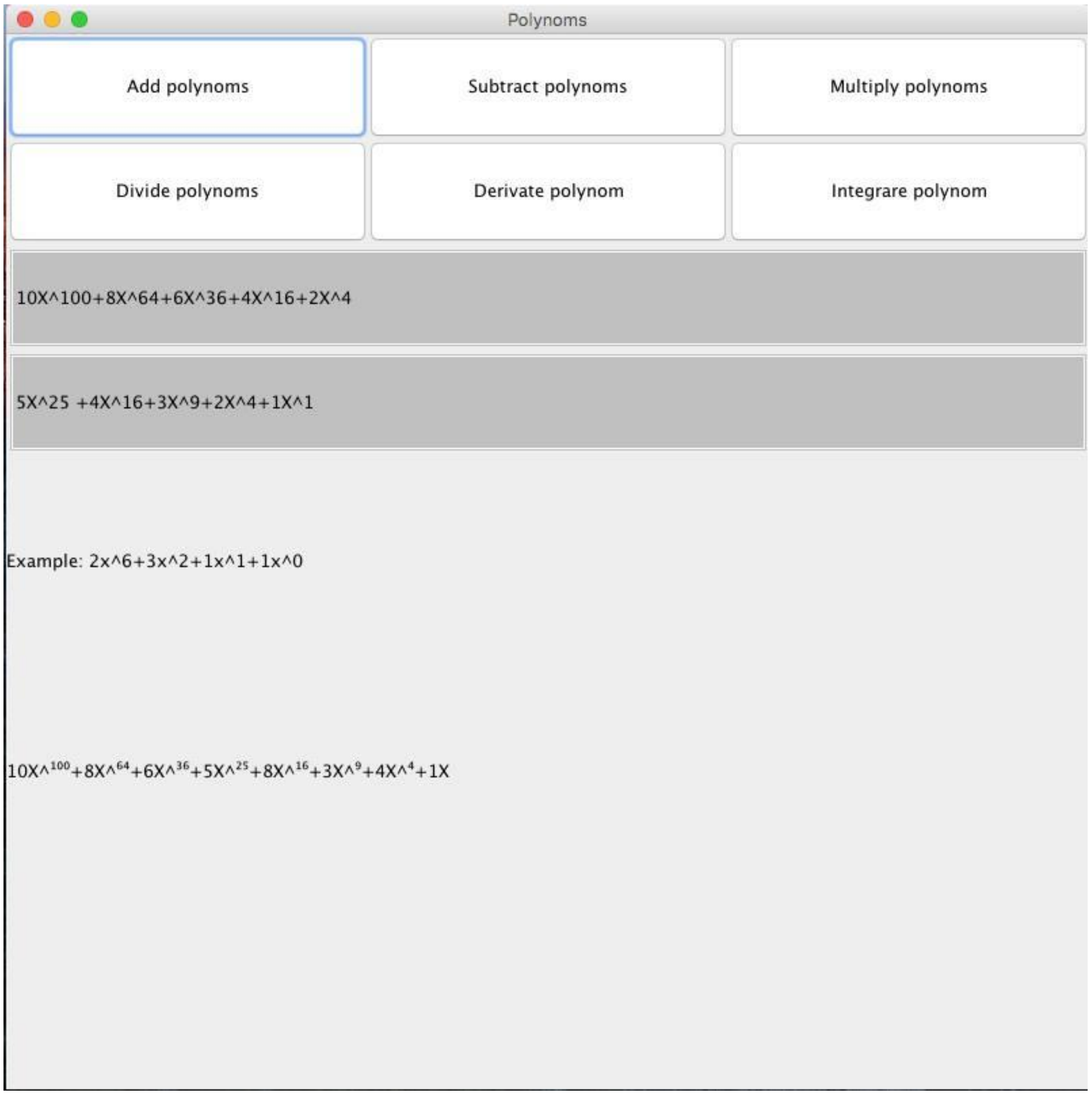
Operations class contains the add, subtract, multiply, divide, derivate, integrate operations. The algorithms that were used for these operations were explained in the Algorithm section. The operations is a little more complex by choosing to store only the given terms.



<<Java Class>>
Operations
Operations()
addPolynoms(Polynom,Polynom):Polynom
subPolynoms(Polynom,Polynom):Polynom
mulPolynoms(Polynom,Polynom):Polynom
divPolynoms(Polynom,Polynom):DivisionPolynoms
derivatePolynom(Polynom):Polynom
integratePolynom(Polynom):Polynom

3.5. User Interface

This application is a user oriented application, so it is important to design an interface that is user friendly. Taking this into consideration, the interface should be intuitive and simple to use. The User Interface was built entirely with Java Swing elements: JFrame with a GridLayout and JPanels added to this frame. Buttons, labels and text field were added on thes panels to make the interface more readable for the user. The size of the window is set to 800/800 pixels.



10X^100+8X^64+6X^36+4X^16+2X^4

5X^25 +4X^16+3X^9+2X^4+1X^1

Example: 2x^6+3x^2+1x^1+1x^0

$10X\wedge^{100}+8X\wedge^{64}+6X\wedge^{36}+5X\wedge^{25}+8X\wedge^{16}+3X\wedge^{9}+4X\wedge^{4}+1X$

The main windows is divided into 4 sections: the button sections where are the buttons for thee operations commands, the input section where are 2 text fields for inserting the polynomials, the hint section which contains an example of a valid polynomial and the result section which contains the resulting polynomial after inserting the polynomials and choosing an operation. The class responsible for creating the user interface is UserInterface.

4. Implementation and testing

The application was implemented using Java programming language and Eclipse IDE for Java developers. The project was created using Maven and also included Junit. The design stages were presented in section 2 and the algorithms, data structures, classes and methods used were described in detail in section 3.

Given the requirements of the problem, the application did not require complex data structures. The logic and the implementation should be as simple as possible in order to be able to reuse code. Using Object-Oriented Programming concepts and a Model View Controller approach, the solution for the assignment is simple and every used class handles its purpose very simple. MVC makes the code easier to understand because the model part is formed from the Polynom, Term and DivisionPolynoms classes which handles only the model part of the problem, the view part is formed of the UserInterface class which is used only for displaying the data or receive the data from the user. The "brain" of the application is the AppLogic class which has the role of a controller that manages the view and model data in order to work as a whole.

During and also after the design and implementation of the assignment, the functionality has to be tested. In order to do this, I used System.err.println(String x) function which prints the string on the standard error output. Another tool that I used is the debugger from Eclipse. It le you trace the execution of code line by line helping you finding minor problems in implementation. In order to verify the correctness of the operations, I wrote Junit tests in order to verify the result of every operation.

JUnit helped me to check if the format is like I want. Because the project is based on mathematical operations the testing of operations is a little hardcoded, such that the result polynomial is compared with the result that has gave me on paper. For example the add operation is tested like this:

```java
@Test
public void testAddPolynoms() {
    resultPolynom = Operations.addPolynoms(firstPolynom, secondPolynom);
    assertEquals(resultPolynom.polynomToString(),"10X^100 +8X^64+6X^36+5X^25+8X^16 +3X^9
 +4X^4 +1X".replace(" ", ""));
}
```

5. Results

I consider I have reached a pretty robust implementation of the Polynomial Assignment. It offers a simple interface and the coding behind is clear and robust. The results of this assignment is a working app that make operations on two given polynomials.

The final application let you manage the next operations on polynomials:

One polynomial	Two polynomials
Derivation	Add
Integrate	Subtract
	Multiply
	Divide

6. Conclusions, what has been learned, further developments

The aim of every assignment is to put into practice what have you learned and for a better understanding of learned concepts. This assignment let me revise my knowledge about polynomials, and after thinking about the design of classes and how every piece should work, the implementation was pretty simple.

Also, it gives me a better understanding of the Java programming language, JSwing and new tools that I used such as Maven, Junit and Javadoc. Even the assignment does not require too complex classes design, it was a good exercise to handle them and find the best solution.

The configuration for Maven let me add JUnit as a dependency:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ro.utcn.pt</groupId>
    <artifactId>Lab1</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.4</version>
        </dependency>
    </dependencies>
</project>
```

The documentation written in Javadoc let you navigate through classes and see the methods and attributes of each class. Javadoc is very simple to use, inserting comments and then get an interactive web page to look into.

```
/**
* AppInterfaceButtonEvents.java - AppInterfaceButtonEvents is the connection between AppLogic and UserInterface classes.
*  Here are defined the methods that implements the AppLogic and are called from UserInterface in order to announce the controller that the user pressed some button.
* @author  Vlad Bonta
* @see AppLogic
* @see UserInterface
*/
```

For further developments I could think of various improvements and new features such as:

- Give the possibility to insert more polynomials and let the user choose on which polynomials to make the operations
- Improve the aspect of the user interface
- Improve the parsing method of the polynomial, so that the user can enter the polynomial more easily
- Implement more operations on polynomials and plot the graphic for a polynomial
- Transform the application into a mobile phone application
- Store the results into a file
- Validate the input string
- Present error messages if he polynomial is not correct

7. Bibliography

[1] http://en.wikipedia.org/wiki/Polynomial

[2] http://en.wikipedia.org/wiki/Use_case

[3] http://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial

[4] https://www.youtube.com/watch?v=l6_ghhd7kwQ

http://www.tutorialspoint.com/junit/