Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Department of Computer Science

2016/2017

Programming Techniques

Homework 1

Polynomials Processing

Lecturer: Prof. Ioan Salomie

Teaching Assistant: Claudia Pop

Student: Acu D. Raul-Mihai

Group: 30425

‘

Table of Contents

1. Problem specification

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

a. System`s requirements

Requirements for accepting the homework:

- Encapsulation
- Classes of at most 300 lines
- Methods of at most 30 lines
- Java naming conventions

For grade "5":

- Graphic User Interface
- Addition and Subtraction operations
- Documentation

For higher grade:

- Multiplication 1p
- Division 1p
- Differentiation 0.5p
- Integration 0.5p
- OOP Design 1p:
  - Polynomial, Monomial classes
  - Using List<> instead of array[]
  - Using foreach instead of for(int i=0…)
- JUnit Tests 1p

b. System`s description

The system provides an Graphical User Interface (GUI) which can be used for processing two polynomials given by the user. This can be done by using the implemented buttons for operations, such as: Addition, Subtraction, Multiplication, Division, Differentiation and Integration, which will be described below.

2. Example of functionality

As we already know, every Polynomial is made of one or more Monomials, each one having a specified coefficient, and a degree.
Denote the Monomial as: (coefficient)x^degree;
Let us have two polynomials: $P_1(x)$ = 17x^2 + 3x
$P_2(x)$ = x + 1

a. Addition

The addition operation consists of grouping the Monomials into a single Polynomial, and, if the case, adding the coefficients of the Monomials that have the same degree.
$P_1(x)$ + $P_2(x)$ = 17x^2 + 4x +1

b. Subtraction

The subtraction operation consists of grouping the Monomials from the first Polynomial, along with the negated ones of the second Polynomial, into a single Polynomial, and, if the case, subtracting the coefficients of the Monomials that have the same degree.
$P_1(x)$ - $P_2(x)$ = 17x^2 + 2x – 1

c. Multiplication

The multiplication operation consists of multiplying each Monomial from the first Polynomial, with each one from the second Polynomial. This is done by multiplying monomial`s coefficients and summing their grades.
$P_1(x)P_2(x)$ = (17x^2 + 3x)(x+1) = 17x^3 + 3x^2 + 17x^2 + 3x = 17x^3 + 20x^2 +3x

d. Division

For the division operation, an auxiliary polynomial is required, and the process of division is done by multiplying the divisor with a Monomial, with the coefficient equal to the division of the first monomial`s coefficient from each polynomial, and with the degree equal to the subtraction of those two degrees, until the degree of the auxiliary polynomial is less than the divisor`s.
$P_1(x)$ / $P_2(x)$ = 17x – 14, remainder of 14.

e. Differentiation

The derivative of the Polynomial is obtained by differentiating each Monomial, that is a new Monomial of coefficient equal to that Monomial`s coefficient multiplied by the degree, and the degree of the new Monomial decreases by 1.
$P_1(x)'$ = 34x + 3

f. Integration

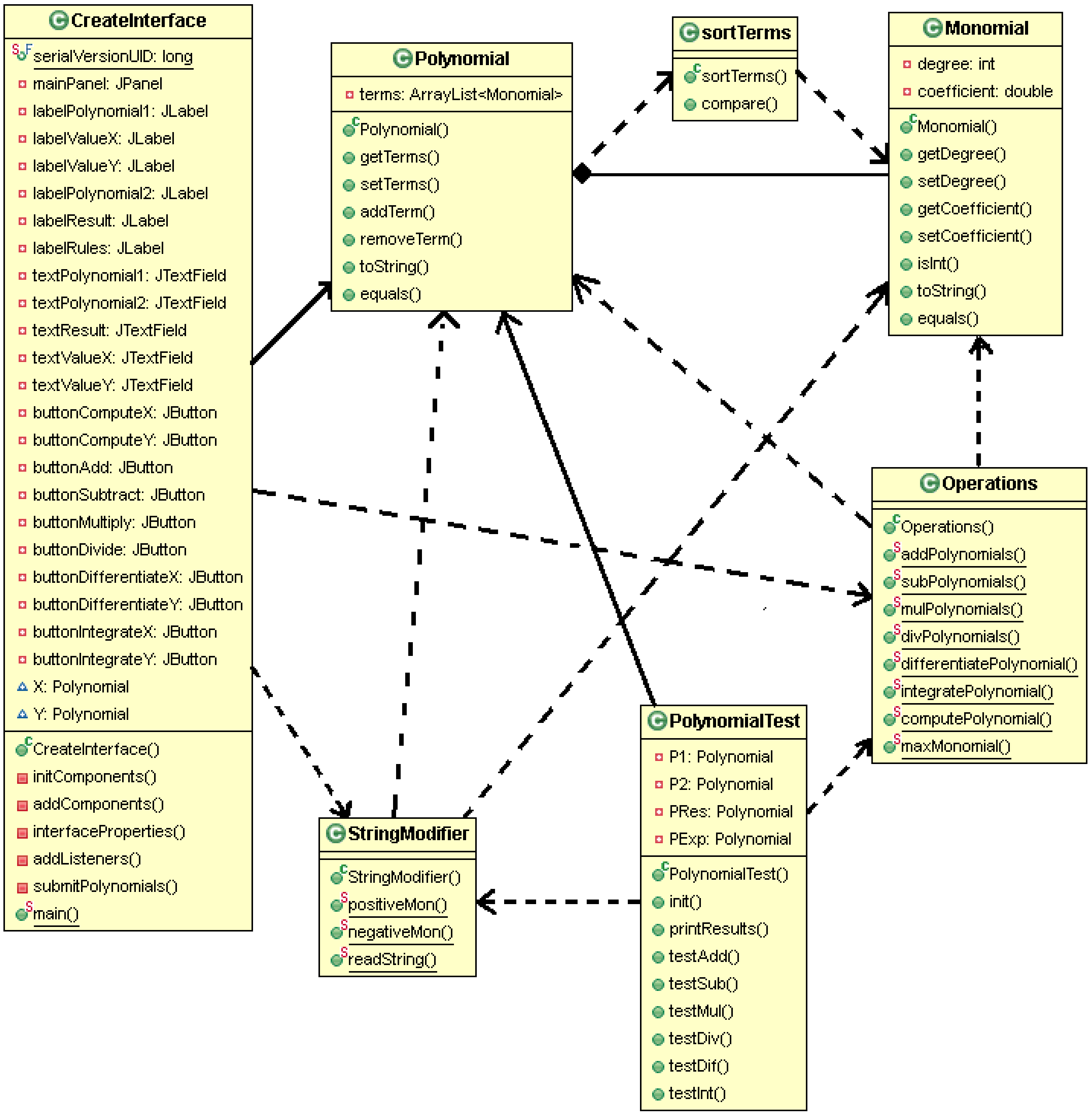
The integration of a Polynomial is obtained by dividing each Monomial`s coefficient by grade + 1, and increasing the Monomial`s grade by one.
$\int P_1(x) =$ 17/3x^3 + 3/2x^2

3. Design

a. UML Class Diagram

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems



b. Packages, classes and methods

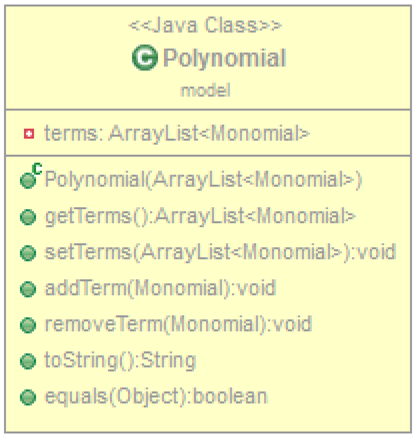
This system uses three packages, each used for different purposes:

- **logic** package – for the JUnit testing
- **model** package – containing the main classes used for Polynomial processing
- **ui** package – providing the graphical user interface resources.

We will start by describing the most important classes, in the **model** package, the other packages being approached later on.

Polynomial class provides :

- the implementation of an object named Polynomial, created with the help of the **Polynomial(ArrayList<Monomial>)** constructor.
- methods that can be used to store, access or modify contents of the Polynomial object. Such methods are enumerated below:

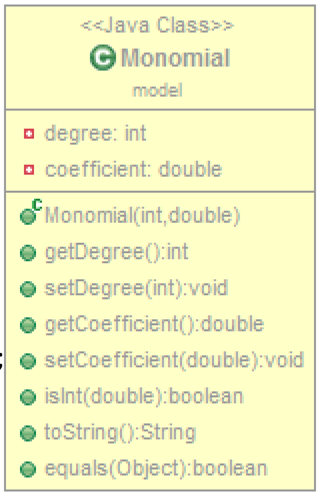

<<Java Class>>
Polynomial
model
terms: ArrayList<Monomial>
Polynomial(ArrayList<Monomial>)
getTerms():ArrayList<Monomial>
setTerms(ArrayList<Monomial>):void
addTerm(Monomial):void
removeTerm(Monomial):void
toString():String
equals(Object):boolean

○ **getTerms()** – returns the elements of the ArrayList<Monomial> terms;
○ **setTerms(ArrayList<Monomials>)** – replaces or sets a new set of elements;
○ **addTerm(Monomial)** – checks if there already exists a Monomial with the same degree, and sums their coefficient, if not, it adds it to the Polynomial;
○ **removeTerm(Monomial)** – removes the Monomial from the Polynomial;
○ **toString()** – outputs the elements of the Polynomial as a single String;
○ **equals(Object)** – this is an override of the equals() method already implemented in java, for comparing the elements one by one, and not their location in memory.

- the basic operand for the Operations class (which will be presented later).

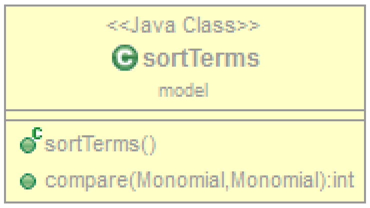
Monomial class provides:

- the implementation of an object named Monomial, created with a **Monomial(int, double)** constructor.
- fields of the Monomial object, such as:
  - **private degree** – of type int.
  - **coefficient** – of type double.
- methods that contribute to the Monomial`s properties:
  - **getDegree()**, **getCoefficient()** – the only way to access Monomial`s fields;
  - **setDegree()**, **setCoefficient()** – set the values of Monomial`s fields;
  - **isInt(double)** – checks if a given value is an integer or a double;
  - **toString()** – outputs the properties of the Monomial as a single String, "(coefficient)x^degree"
  - **equals(Object)** – has the same purpose as the one in the Polynomial class;
- the only type of Object accepted in the Polynomial`s structure.



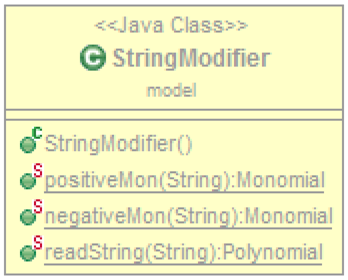
SortTerms class :

- implements the Comparator interface, is used for sorting the elements (Monomials) of the Polynomial;
- has only a method, **compare(Monomial,Monomial)** that is an override of the existing compare method in Java.



StringModifier class is highly important due to it`s function:

- the only way to store the values Polynomial received from the Graphical User Interface.
- it`s methods receive an input String, that will be split in substrings, each of them containing a Monomial. The split is realized based on the "+" and "-" characters, setting the Monomial`s coefficients signs:
  - **readString(String)** – is the main method here, and it uses both of the *positiveMon(String)* and *negativeMon()* methods. It basically receives the input string, doing a split based on sign. If the substring starts with "+" it will call the positiveMon(String) method and if it starts with "-" the *negativeMon(String)* will be called. Their behavior is explained below:
  - **positiveMon(String)** – receives the substring generated and sets the attributes of the Monomial accordingly, creating a positive Monomial then adding it to the Polynomial;
  - **negativeMon(String)** – has similar behavior as the *positiveMon(String)*, but it adds a negative Monomial to the Polynomial.

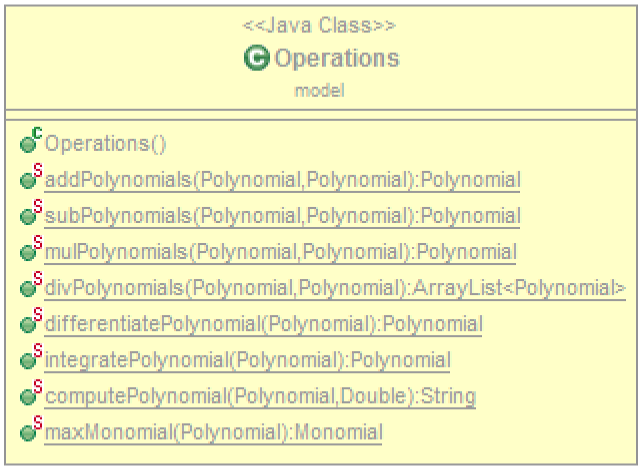


<<Java Class>>
StringModifier
model
StringModifier()
positiveMon(String):Monomial
negativeMon(String):Monomial
readString(String):Polynomial

Operations class is the main provider for the User, most of it`s methods being called directly by the Graphical User Interface. These methods receive as parameters, one or two Polynomials, depending on the requiements:

- **addPolynomials(Polynomial,Polynomial)**

Computes the addition between two given Polynomials;
<u>*Algorithm description*</u>*:* an auxiliary Polynomial is created, in which both of the polynomials are stored, then by using two foreach loops, for iterating through the Polynomials; If there are two Monomials, with equal degrees, their coefficients are summed, and both Monomials removed from the auxiliary Polynomial, returning the expected result;



<<Java Class>>
Operations
model
Operations()
addPolynomials(Polynomial,Polynomial):Polynomial
subPolynomials(Polynomial,Polynomial):Polynomial
mulPolynomials(Polynomial,Polynomial):Polynomial
divPolynomials(Polynomial,Polynomial):ArrayList<Polynomial>
differentiatePolynomial(Polynomial):Polynomial
integratePolynomial(Polynomial):Polynomial
computePolynomial(Polynomial,Double):String
maxMonomial(Polynomial):Monomial

- **subPolynomials(Polynomial,Polynomial)**

Computes the subtraction of the second Polynomial, from the first one.
<u>*Algorithm description*</u>*:* a copy of the second Polynomial is created, on which we use a foreach loop, for inverting each Monomial`s coefficient, then performing the addition of the first Polynomial and the copy.

- **mulPolynomials(Polynomial,Polynomial)**

Computes the multiplication of two given Polynomials (in that order).
<u>*Algorithm description*</u>*:* we perform the ordinary multiplication of Monomials (see **2.c**), but this will result in an unordered Polynomial, having more Monomials with the same degree. We can fix this, by summing the coefficients of the Monomials of equal degrees, while removing them and storing in the Polynomial only the resulted Monomial;

- **divPolynomials(Polynomial,Polynomial)**

Computes the division of the first Polynomial, by the second Polynomial. This operation implies every operation already mentioned. This method returns an ArrayList<Polynomial>, in which is stored the
Algorithm description: For this method to be implemented correctly, we must create 3 new Polynomials: One for the Remainder, one for the Result and an auxiliary one, which will be modified and then cleared at every step. At first, the result will be given by dividing the highest degree Monomials from each Polynomial. After that, the auxiliary Polynomial is set to be equal to the Result, then, multiplied by the second Polynomial, and subtracting it from the first Polynomial. This algorithm is repeated until the resulted Polynomial`s max degree is less than the second one`s. What`s left, is stored in the Remainder Polynomial and added along with the Result, to the ArrayList<Polynomial>;

- **differentiatePolynomial(Polynomial)**

Computes the derivative of the Polynomial received as a parameter.
Algorithm description: Using a foreach loop for iterating through the Polynomial, the method adds a new Monomial with the coefficient multiplied by the degree, and the degree increased by one, to an auxiliary Polynomial, which will be returned as the result;

- **integratePolynomial(Polynomial)**

Computes the inverse of differentiation, called integration of the given Polynomial.
Algorithm description: Using a foreach loop for iterating through the Polynomial, the methods creates a new Monomial with the coefficient divided by (degree + 1), and the degree decreased by one, then adds it to an auxiliary Polynomial which will be returned as the result;

- **computePolynomial(Polynomial,Double)**

Computes the value of the Polynomial, on a given point.
Algorithm description: At first, we initialize a new Double value, on which we will compute the result. Using a foreach loop for iterating through the Polynomial, at each Monomial, the coefficient is multiplied by value at power degree of the Monomial, and added to the previous value;

- **maxMonomial(Polynomial)**

Returns the max degree of the Polynomial, used in the division process.

4. Using and testing the system

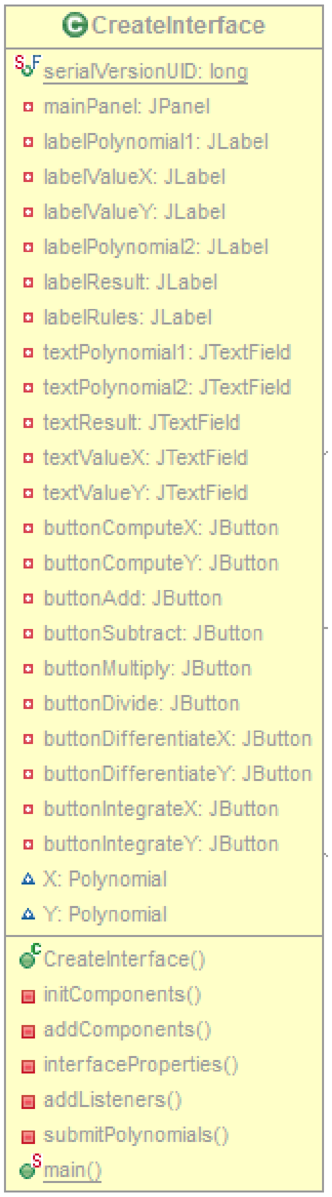
a. User interface

The graphical user interface (GUI), is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.

Designing the visual composition and temporal behavior of a GUI is an important part of software application programming in the area of human–computer interaction. Its goal is to enhance the efficiency and ease of use for the underlying logical design of a stored program.

Such an user interface, is implemented in the **ui** package, by using the **CreateInterface** class. The user interface is based on the properties of the above mentioned packages. All the objects we need are declared as attributes of the class and they are initialized in the constructor **CreateInterface()**.

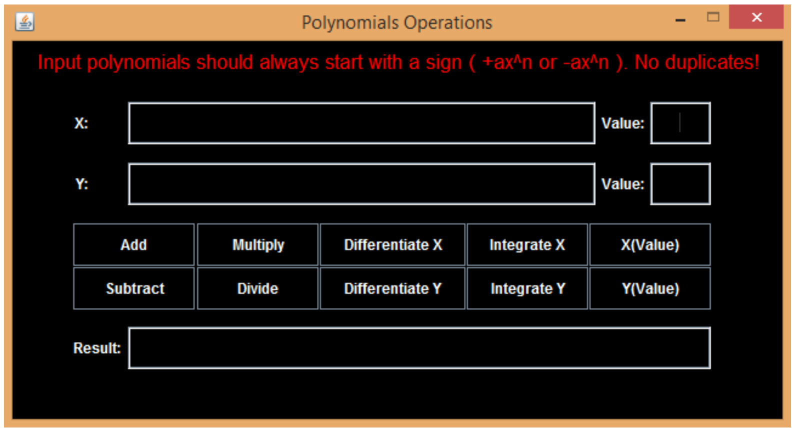
- **JPanel** for the main window application:
  - *mainPanel;*
- **JLabels** for the written text:
  - *labelPolynomial1;*
  - *labelPolynomial2;*
  - *labelResult;*
  - *labelRules;*
- **JTextFields** used to read the polynomials or print the result:
  - *textPolynomial1;*
  - *textPolynomial2;*
  - *textResult;*
- **JButtons** for applying the desired operation.
  - *buttonAdd;*
  - *buttonSubtract;*
  - *buttonMultiply;*
  - *buttonDivide;*
  - *buttonDifferentiateX;*
  - *buttonDifferentiateY;*
  - *buttonIntegrateX;*
  - *buttonIntegrateY;*
- **Various methods** that contribute to the element`s properties, are responsible of adding them to the interface and describing their behavior:
  - *interfaceProperties()* – sets the Location, Size and Title of the JFrame.
  - *initComponents()* – sets the properties of the elements described before;
  - *addComponents()* – adds the elements to the main window application;
  - *addListeners()* – assigns each button the according action;
  - *submitPolynomials()* – performs the conversion of the Strings read from the text fields, into Polynomials;



The actual **Graphical User Interface** is presented below.

Every button represented in the GUI, calls the *submitPolynomials()* method, for setting the values of the Polynomials to be computed, while performing the according operation.

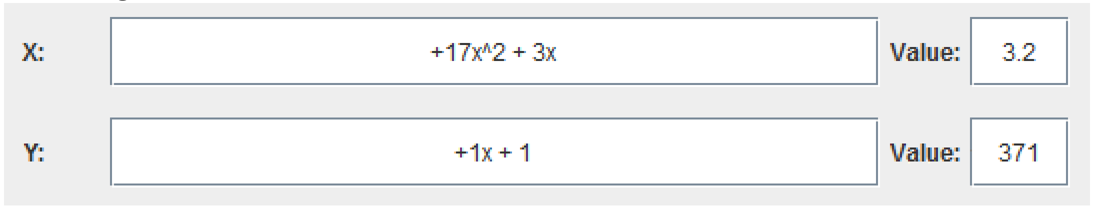
- **Add** – performs the addition of X and Y;
- **Subtract** – subtracts Y from X;
- **Multiply** – multiplies X by Y;
- **Divide** – divides X by Y;
- **Differentiate X** – computes the derivative of X;
- **Differentiate Y** – computes the derivative of Y;
- **Integrate X** – performs the integration of X;
- **Integrate Y** – performs the integration of Y;


Polynomials Operations

Input polynomials should always start with a sign (+ax^n or -ax^n). No duplicates!

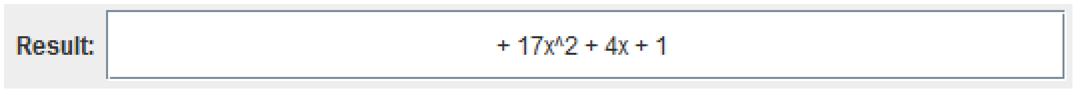
b. Testing the functionality of the interface

For this, we will choose the same two Polynomials for every operation, observing the Interface`s response according to the desired action.

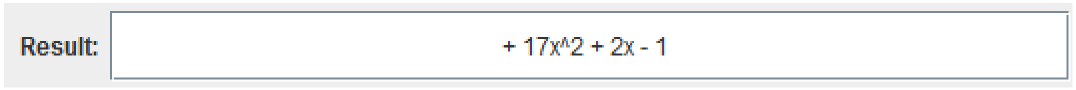

X: +17x^2 + 3x Value: 3.2

Y: +1x + 1 Value: 371

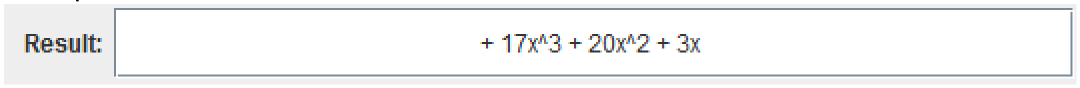
Addition:


Result: + 17x^2 + 4x + 1

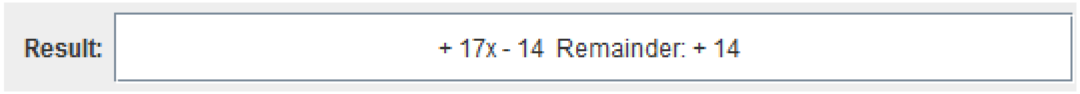
Subtraction:


Result: + 17x^2 + 2x - 1

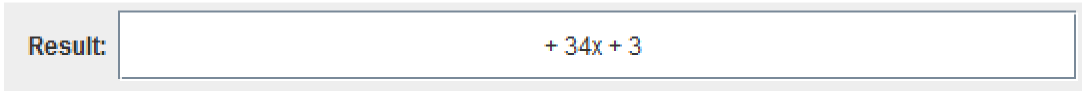
Multiplication:


Result: + 17x^3 + 20x^2 + 3x

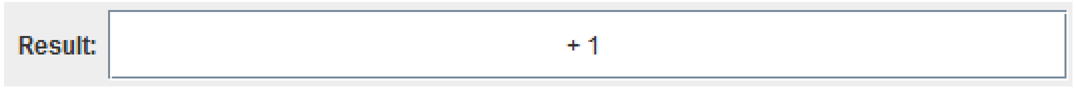
Division:


Result: + 17x - 14 Remainder: + 14

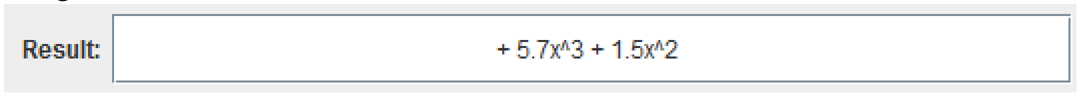
Differentiation of X:


Result: + 34x + 3

Differentiation of Y:

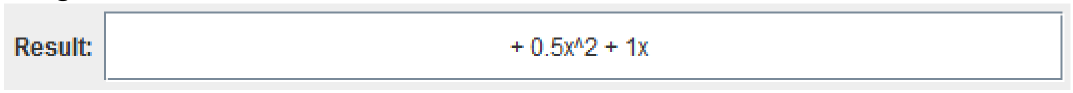

Result: + 1

Integration of X:



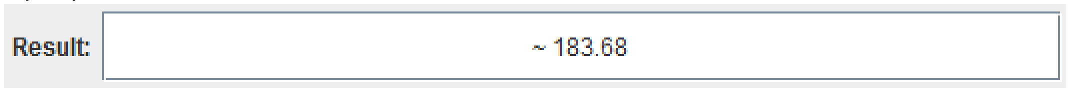
Result: + 5.7x^3 + 1.5x^2

Integration of Y:



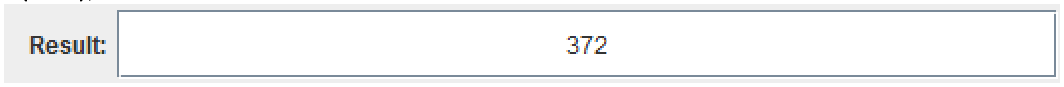
Result: + 0.5x^2 + 1x

X(3.2):



Result: ~ 183.68

Y(371);



Result: 372

c. JUnit testing

JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit, that originated with JUnit.

JUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

Testing is the process of checking the functionality of an application to ensure it runs as per requirements. Unit testing comes into picture at the developers' level; it is the testing of single entity (class or method). Unit testing plays a critical role in helping a software company deliver quality products to its customers.

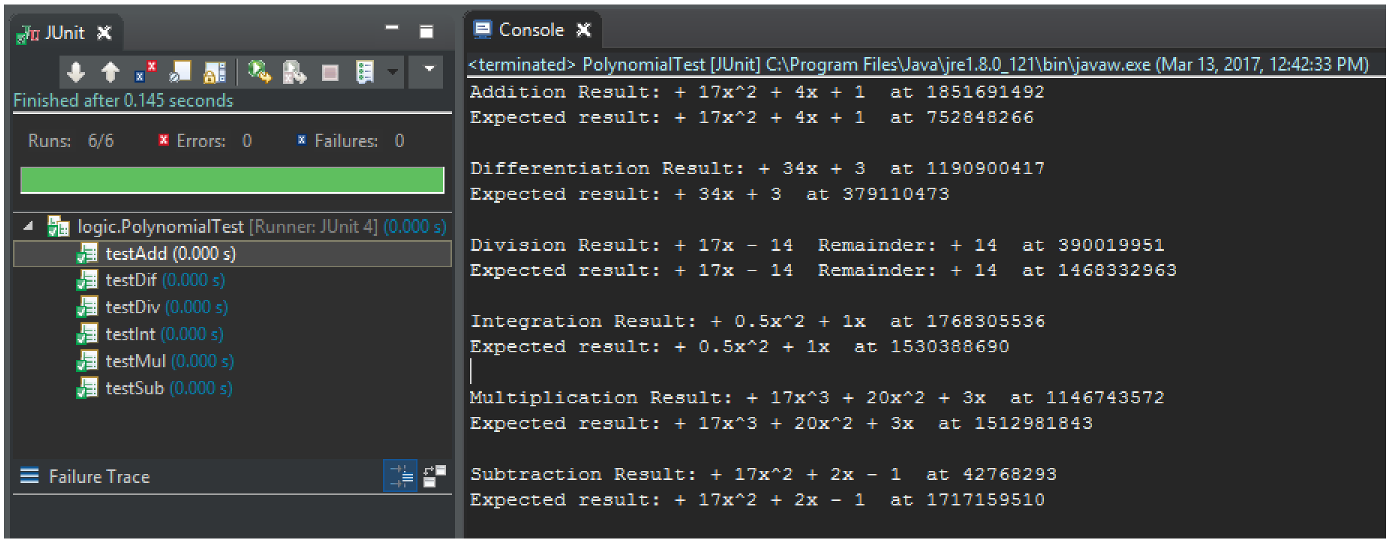
Using JUnit testing, we will evaluate the functionality of the methods included in the Operations class, but also the behavior of the objects Polynomial and Monomial.

The process is described as:

- Creating a new JUnit Test case that has the same format as an ordinary java class file (.java), but it does not have a constructor, although fields can be assigned to it.
- Implementation of desired testing methods
  - testAdd() for testing the Operations.addPolynomials(Polynomial,Polynomial);
  - testSub() for subtraction;
  - testMul() for multiplication;
  - testDiv() for division;
  - testDif() for differentiation;
  - testInt() for integration;

In each of the methods mentioned before, we set the Polynomials to the desired values, and as well, creating an auxiliary Polynomial, which we set to the Expected Value. Then, we perform an assertEqual() method that will mark the test as succeeded, if the expected value equals to the resulted one, or as failure if otherwise. The assertEqual() calls the override of the equals() method from both Polynomial and Monomial classes, comparing each Polynomial`s value, not their addresses.

The results of the JUnit test are presented below:



```
<terminated> PolynomialTest [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Mar 13, 2017, 12:42:33 PM)
Addition Result: + 17x^2 + 4x + 1  at 1851691492
Expected result: + 17x^2 + 4x + 1  at 752848266

Differentiation Result: + 34x + 3  at 1190900417
Expected result: + 34x + 3  at 379110473

Division Result: + 17x - 14  Remainder: + 14  at 390019951
Expected result: + 17x - 14  Remainder: + 14  at 1468332963

Integration Result: + 0.5x^2 + 1x  at 1768305536
Expected result: + 0.5x^2 + 1x  at 1530388690

Multiplication Result: + 17x^3 + 20x^2 + 3x  at 1146743572
Expected result: + 17x^3 + 20x^2 + 3x  at 1512981843

Subtraction Result: + 17x^2 + 2x - 1  at 42768293
Expected result: + 17x^2 + 2x - 1  at 1717159510
```

5. Conclusion

a. Advantages and Disadvantages

➢ **Advantages**

- Provides a faster and easier way of processing the Polynomials, rather than the manual computation of each operation;
- Easy to use;
- Plain and simple, nothing to be confused by.

➢ **Disadvantages**

- Has some restrictions when writing the input Polynomials;
- It might fail if given wrong inputs;
- Has weird functionality when the Polynomial input has duplicates.

b. Future development: ways of improving the system

- Designing a better algorithm for performing the operations;
- Trying to use as less memory as possible;
- Trying to improve the response time;
- Designing a more user-interactive interface;
- Modifying the input methods of the polynomials, so the values would not be interpreted wrongly;