Ian Chelaru
Technical University of Cluj-Napoca
2nd year Computer Science
Programming Techniques 2017/2018

Homework 1 – Specification

1. Homework's objective

The main purpose of the homework is to design and implement a system to process polynomials of one variable and integer coefficients. The application is written in java programming language. The integrated development environment (IDE) used is Eclipse.

Secondary objectives:

- implement the following operations: addition, subtraction, multiplication, division of two polynomials and differentiation, integration of one polynomial;
- implement a graphical user interface (GUI) which allows the user to perform the operations

2. Features

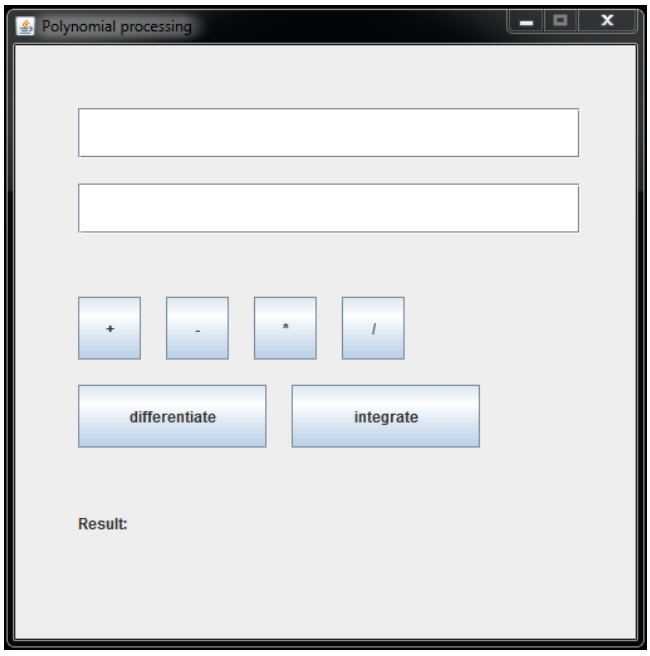

Figure 1 - GUI

When running the application, a GUI shows on the screen (Figure 1). The GUI has two text fields for the input polynomials, six buttons which perform the desired operations and a label for the output polynomial.

For the addition, subtraction, multiplication and division the user has to perform the following steps:

- In the first text field, the first polynomial should be given. The input format of the polynomial consists in an enumeration of the coefficients (given in ascending order with respect to the polynomial powers). The missing terms have the coefficient equal to 0. The 0 coefficients also have to be specified in the enumeration. Each coefficient is separated from the previous one by a space.
  For example, polynomial -2x^4+3x^2–x should be given at the input in the following format:
  0 -1 3 0 -2
  first 0 represents the coefficient of x^0 (missing term);
  -1 is the coefficient of x^1;
  3 is the coefficient of x^2;
  second 0 is the coefficient of x^3 (missing term);
  -2 is the coefficient of x^4.

- In the second text field, the second polynomial should be given. The input format is the same as the one described before.
- After both input polynomials are given, the desired operation is done by clicking on one of the first four buttons. Each button correspond to one operation:
  "+" – performs the addition of the two polynomials given as inputs (Figure 2)
  "-" – performs the subtraction of the second polynomial from the first polynomial
  "*" – performs the multiplication of the two polynomials
  "/" – performs the division of the first polynomial by the second polynomial
- The output polynomial will show at the bottom of the frame, under the "Result:" text. The output format of the polynomial is the following:

  c1+c2x+c3x^2+…+cnx^n

  where c1, c2, c3, …, cn are the coefficients of the polynomial and n is the degree of the polynomial.

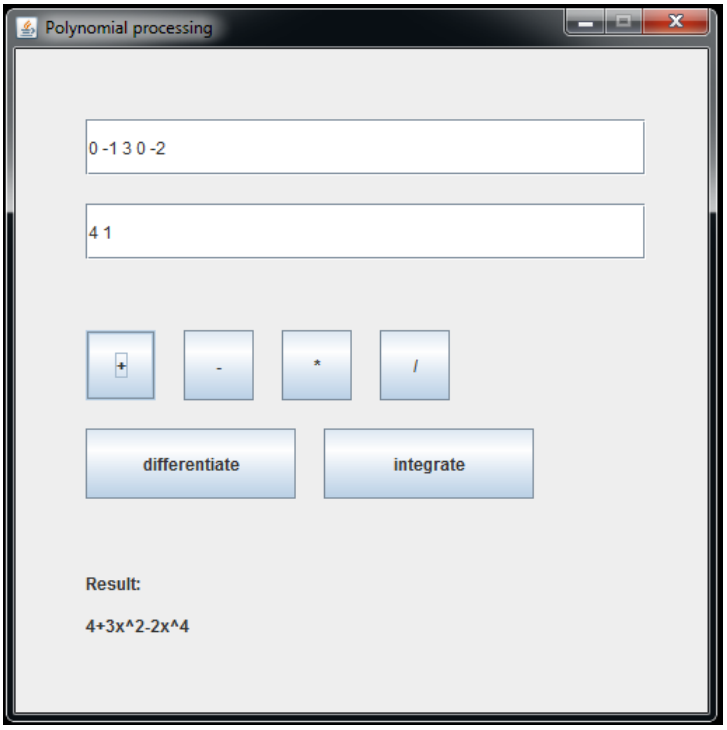

Figure 2 - Addition

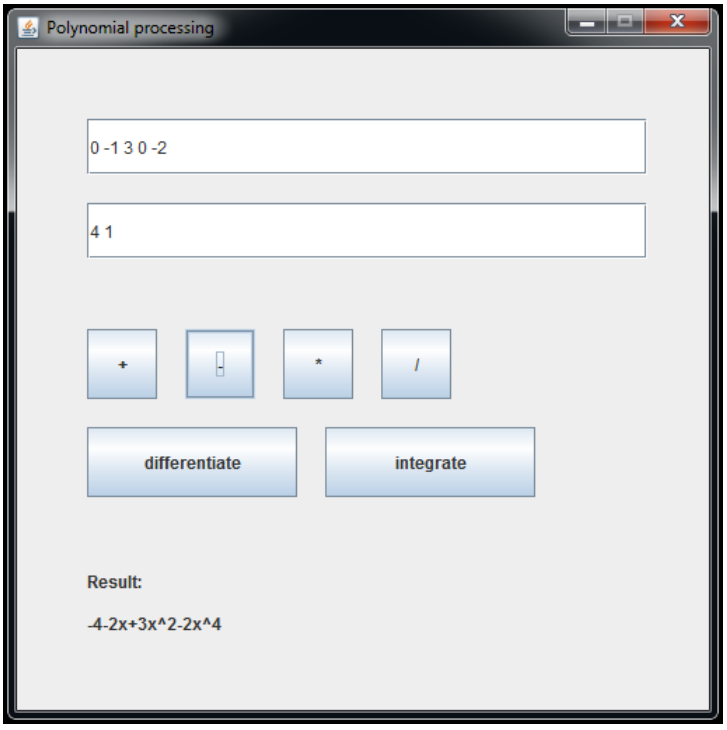

Figure 3 - Subtraction

Figure 4 - Multiplication

Figure 5 - Division

Figure 2 shows the addition of the polynomials –x+3x^2-2x^4 and 4+x. The sum is equal to –x+3x^2-2x^4+4+x = 4+3x^2-2x^4 as the result shows.

Remark: Consider p1 to be the polynomial written in the first text field and p2 the polynomial written in the second text field. In order to compute p2-p1 or p2/p1, the user has to interchange the contents of the text fields.

For the differentiation and integration operation, the user has to perform the following steps:

- The polynomial which has to be differentiated or integrated must be written in the first text field. The format is the one presented above.
- Click on one of the two last buttons:
  "differentiate" - differentiate the polynomial
  "integrate" – integrate the polynomial
- The output polynomial will show at the bottom of the frame, under the "Result:" text.

Remark: If a division or integration operation is performed the real coefficients of the output polynomial will be written with at most two decimals, while the integer coefficients will be written with no decimals.

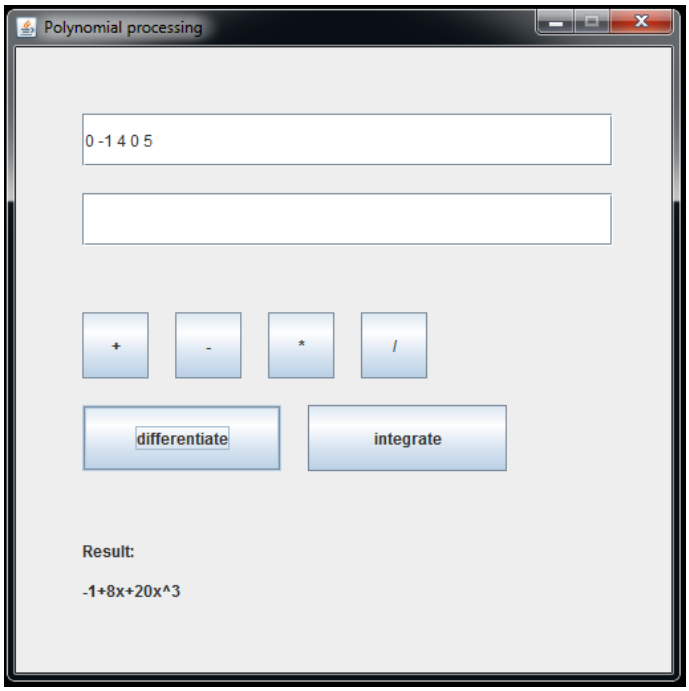

Figure 6 - Differentiation

Figure 7 - Integration

After an operation is done, the text fields which contain the input polynomials remain unchanged. Thus, the user can perform more operations on the same input without rewriting the input more times.

In order to exit the application click the “x” button on the top right of the frame.

Error messages:

The input coefficients have to be integer numbers. A single space is the only delimiter allowed. If the text fields contain characters other than digits and spaces the application will signal that the input format is incorrect. The warning is done when pressing one of the buttons. Instead of returning a polynomial the application will print an error message (“Invalid input”). If there are more spaces between two consecutive coefficients the same error will occur.

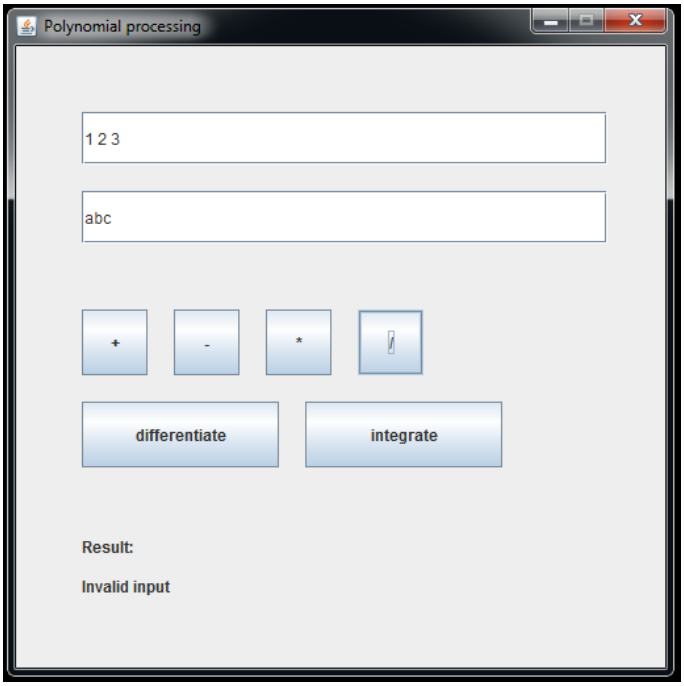

Figure 8 – Invalid input

If the user tries to divide by the 0 polynomial, an error message will occur (“Division by 0”).

Figure 9 – Division by 0

Errors:

An error which may occur during the usage of the application is the incomplete displaying of the output polynomial. If the result has too many terms, even though the polynomial has been computed correctly, only the first terms will be displayed. This is because of the limited size of the labels. The text "…" will show at the end of the result if the length of the string is greater than the length of the label (Figure 10).

Figure 10

The size of the input polynomials is also restricted to the length of the text fields.

If the input or output coefficients get out of the integer type range (-21474836 - 21474836), wrong results will be displayed. In figure 11, it is performed the addition between 21474836 and 1. The expected result is 21474837. However, because this value exceeds the integer range a wrong calculus is performed and the effective result is not equal to the correct one.

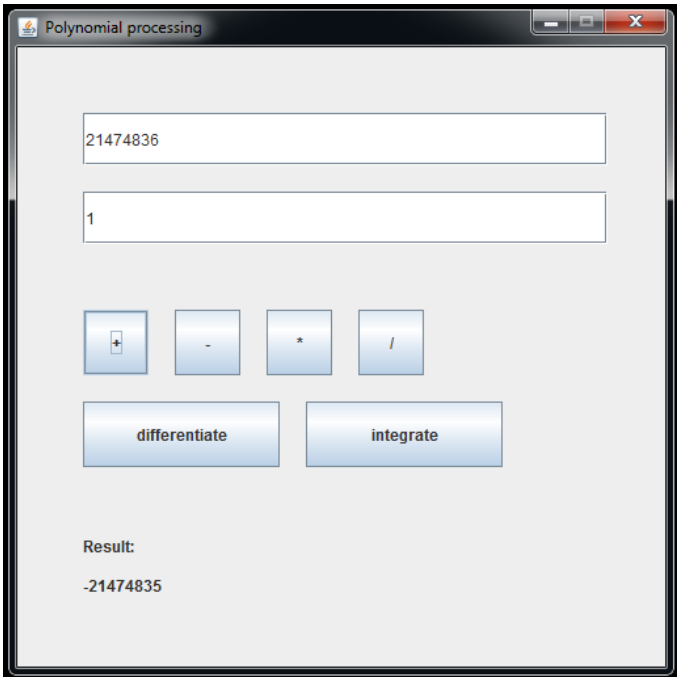

Figure 11 – Out of range

Assumptions:

In order for the application to work properly, it is assumed that the input is given in a manner in which it will not generate one of the previous described unresolved errors.

Remark: Another error which is not handled but it does not affect the system behavior is the presence of one or more zeros in front of a coefficient (for instance 000555 and 555 represent the same coefficient – Figure 12).

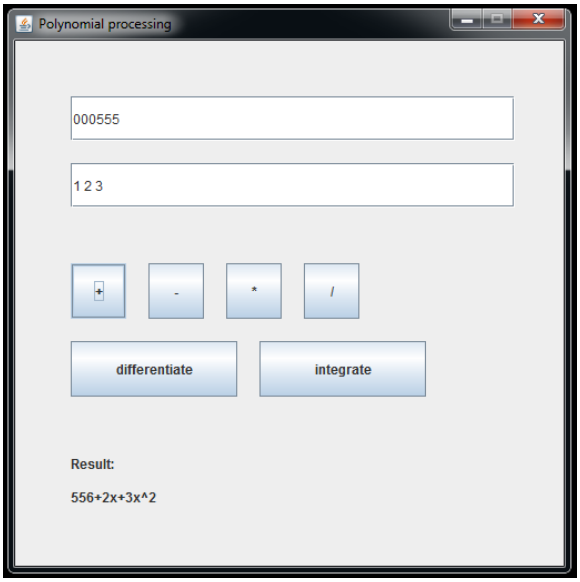

Figure 12

3. Design

In order to design the system, the MVC (model – view – controller) pattern is used. This pattern splits the application into three parts. The model consists in implementing classes for the objects used to solve the problem. These classes have the purpose to manage the input data to implement the methods which solve the application requirements. The view creates the GUI which helps the user to communicate with the application. The controller takes the inputs received by the view, send it to the model (which performs the computations), takes the result and sends it to the user (using the view). The model and the view do not communicate directly. They communicate only through the controller.

Model

The application main requirement is to perform polynomial operations. Thus, a class Polynomial is needed. The Polynomial class should behave like a polynomial from the real world. A polynomial is well determined by its degree and its list of coefficients. So the Polynomial class should have as the attributes a list of integer coefficients and an integer degree .In order to have a more modular structure, a Monomial class is created. In real world, a monomial is a polynomial with only one term. Thus, the Monomial class will be determined by degree and coefficient.

Using the Monomial class, the Polynomial class can be implemented as a list of Monomial objects. Between the Monomial class and the Polynomial class is an aggregation relation.

The input polynomials have integer coefficients. However, the result of the division and integration operations could be a polynomial which has both real and integer coefficients. In order to handle this requirement, two new classes are created: IntegerMonomial and RealMonomial. Both of them are subclasses of the Monomial class (they inherit all the attributes and methods of the superclass). The difference between them is that the coefficients of the IntegerMonomial objects are of type integer and the coefficients of the RealMonomial objects are of type double.

Having these four classes (Polynomial, Monomial, IntegerMonomial, RealMonomial), the model can be created.

View

The GUI has only one frame, so a single class named View is sufficient to implement this part of the system.

Controller

Considering that the design consists in one model and one view which have to communicate data one to another, a class called Controller is enough to perform this task.

The project is structured on three packages: models, controllers and views. Each one contains the classes which handle the corresponding part of the application (Figure 13).

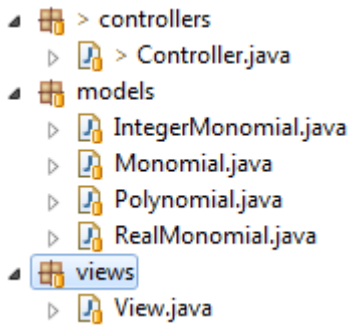

Figure 13

UML diagram

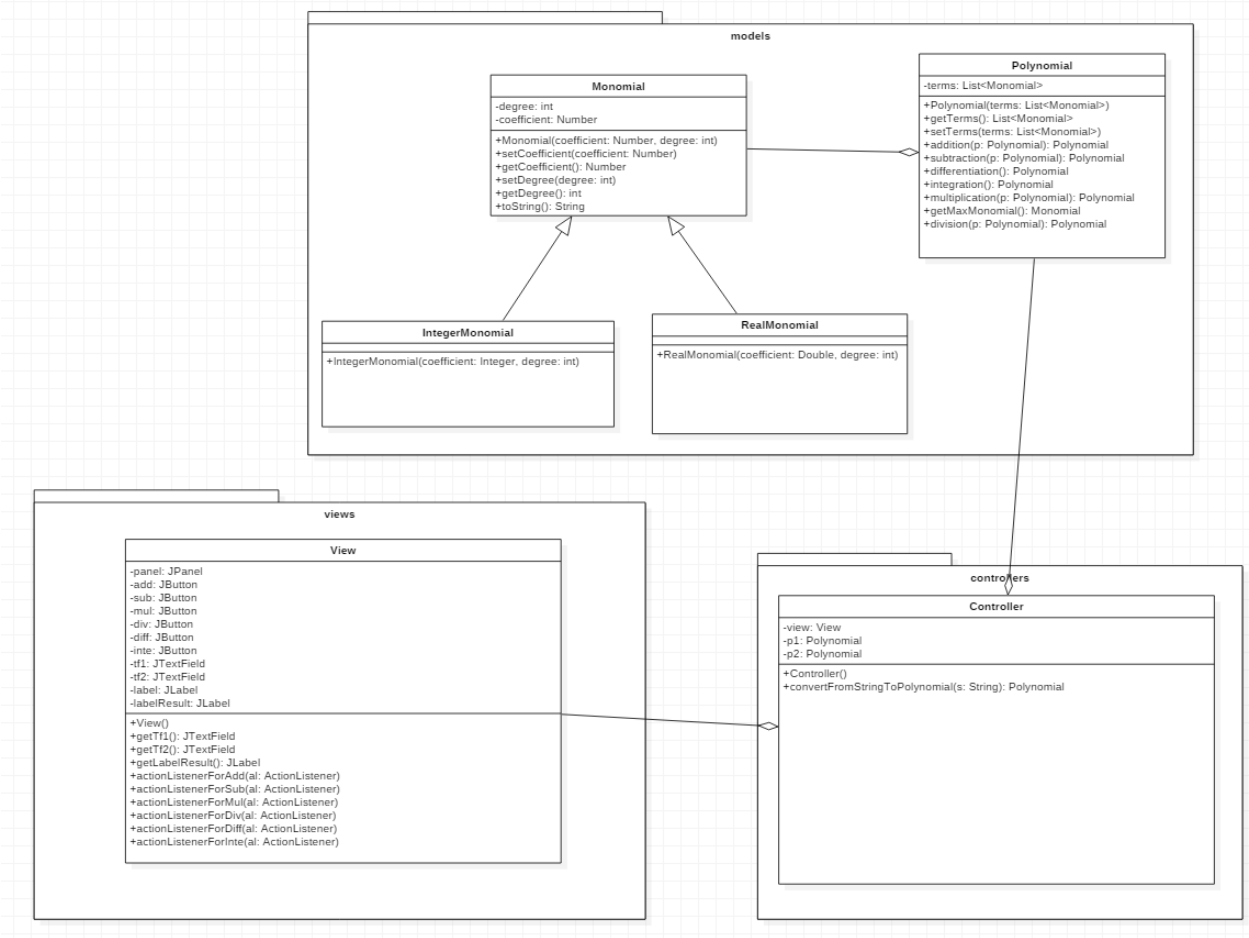

Figure 14 - UML

Data structures:

The ArrayList class, which is a “resizable-array implementation of the List interface” (javadoc) from the java.util package, is used to handle the list of monomials from the Polynomial class. The ArrayList class has some implemented methods which are used to manage the polynomial’s data:

- iterator
  Iterator<E> iterator()
  “Returns an iterator over the elements in this list in proper sequence” (javadoc)

This is used to access one by one all the elements from the list;

- size
  int size()
  “Returns the number of elements in this list” (javadoc);
- get
  E get(int index)
  “Returns the element at the specified position in this list” (javadoc);
- forEach
  default void forEach(Consumer<? super T> action)
  “Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception” (javadoc)
  This behaves as a for loop.

4. Implementation

Class Monomial

- `public class` **`Monomial`**
- `extends java.lang.Object`

Monomial is a function of variable x. It is perfectly determined by its coefficient and degree. The value of the monomial in the point x is computed by rising x to the power d and multiply the result with c: c*x^d, where c is the coefficient and d the degree.

- ***Constructor Detail***

- **Monomial**

- ```
  public Monomial(java.lang.Number coefficient,
                  int degree)
  ```

Constructs and initializes a monomial.

Parameters:

`coefficient - the coefficient of the newly constructed monomial`

`degree - the degree of the newly constructed monomial`
  ```

- ***Method Detail***

- **getCoefficient**

```
public java.lang.Number getCoefficient()
```

- **setCoefficient**

```
public void setCoefficient(java.lang.Number coefficient)
```

- **getDegree**

```
public int getDegree()
```

- **setDegree**

```
public void setDegree(int degree)
```

- **toString**

```
public java.lang.String toString()
```

Returns a string which represents a monomial object in the way in which it is seen in the real world by any person (cx^d).

Overrides:

```
toString in class java.lang.Object
```

Class IntegerMonomial

- `public class` **`IntegerMonomial`**
- `extends pt2018.assign1.models.Monomial`

IntgerMonomial is a subclass of the Monomial class. The coefficient is an Integer object.

- ***Constructor Detail***

- **IntegerMonomial**

-
```
public IntegerMonomial(java.lang.Integer coefficient,
                       int degree)
```

Constructs and initialize a monomial with integer coefficient. It calls the constructor from the super class Momonial using the keyword "super".

Parameters:

```
coefficient - the coefficient of the newly constructed monomial

degree - degree the degree of the newly constructed monomial
```

Class RealMonomial

- `public class` **`RealMonomial`**
- `extends pt2018.assign1.models.Monomial`

RealMonomial is a subclass of the Monomial class. The coefficient is a Double object.

- ***Constructor Detail***

- **RealMonomial**

- `public RealMonomial(java.lang.Double coefficient,`
  `int degree)`

Constructs and initialize a monomial with real coefficient. It calls the constructor from the super class Momonial using the keyword "super".

Parameters:

`coefficient - the coefficient of the newly constructed monomial`

`degree - degree the degree of the newly constructed monomial`

Class Polynomial

- `public class` **`Polynomial`**
- `extends java.lang.Object`

Polynomial class describes the polynomial function of variable x. A polynomial has one or more term, each term representing a monomial of the same variable x. The value of the polynomial in the point x is obtained by computing all the monomials over the same variable x and adding the results.

- ***Constructor Detail***

- **Polynomial**

`public Polynomial(java.util.List<pt2018.assign1.models.Monomial> terms)`

Construct and initializes a polynomial.

Parameters:

`terms - the list of monomials which form the polynomial`

- ***Method Detail***

- **getTerms**

```
public java.util.List<pt2018.assign1.models.Monomial> getTerms()
```

- **setTerms**

```
public void setTerms(java.util.List<pt2018.assign1.models.Monomial> terms)
```

- **addition**

```
public Polynomial addition(Polynomial p)
```

Computes the addition between this polynomial and the one which is transmitted as parameter. The algorithm add the coefficients of the terms which have the same degree. The terms which do not have a correspondent in the other polynomial are put in the result polynomial at the end.

Parameters:

p - polynomial which is added to the polynomial which calls the method

Returns:

a new polynomial object which is equal to the sum of the two input polynomials

- **subtraction**

```
public Polynomial subtraction(Polynomial p)
```

Computes the subtraction of the polynomial which is transmitted as parameter from this polynomial. The algorithm performs like the one for the addition. In this case the coefficients are subtracted one from another. In the case there are more terms in the in the argument polynomial, the opposite of its terms are added to the result.

Parameters:

p - polynomial which is subtracted from the polynomial which calls the method

Returns:

a new polynomial object which is equal to the difference of the two input polynomials

- **differentiation**

```
public Polynomial differentiation()
```

Differentiate this polynomial. The operation is done by multiplying the degree with the coefficient, the result being the new coefficient. The new degree is computed by subtracting 1 from the old one.

Returns:

a polynomial which is equal to the this polynomial derivative

- **integration**

```
public Polynomial integration()
```

Integrate this polynomial. The new degree is computed by adding 1 to the old one. The new coefficient is obtained by dividing the old coefficient by the new degree.

Returns:

a polynomial which is equal to the this polynomial antiderivative

- **multiplication**

```
public Polynomial multiplication(Polynomial p)
```

Computes the multiplication of this polynomial with the one which is transmitted as parameter. The algorithm multiply each term of this polynomial with each term of parameter polynomial. Every product obtained in this way is converted into a polynomial and is added to the final result.

Parameters:
p - polynomial which is multiplied to the polynomial which calls the method

Returns:
a new polynomial object which is equal to the product of the two input polynomials

- **division**

```
public Polynomial division(Polynomial p)
```

Divides this polynomial by the polynomial transmitted as a parameter. The algorithm applied is the polynomial long division.

Parameters:
p - polynomial which divides this polynomial

Returns:
a new polynomial which represents the quotient

- **toString**

```
public java.lang.String toString()
```

Overrides:
toString in class java.lang.Object

- **getMaxMonomial**

```
public pt2018.assign1.models.Monomial getMaxMonomial()
```

Returns:
the Monomial from the list of Monomials with the biggest degree and coefficient different from 0

GUI

The class which implements the GUI (Figure 1) is called View. It extends the JFrame class and it uses the the most basic components of a frame:

- one JPanel which contains all the components and has an unique layout;
- two JTextFields where the input polynomials are written;
- six JButtons which performs the operations;
- one JLabel (named “label”) with the text “Result” to simply make the user aware of where the result should occur;
- one JLabel (named “labelResult”) with no initial text; when performing an operation, the result or the error message will be written here.

All these components are private attributes of the class. The “labelResult” and the two text fields are the only attributes for which there exists a getter method (one for each). This is because the controller only needs to have access to these attributes in order to perform its task.

The constructor View() is a public one, with no parameters. Inside the constructor, all the components are added to the panel, which is added to the frame. For each attribute, the method setBounds(x, y, width, height) is called in order to place it on the panel.

Besides the three getters implemented, the class has one method for each button which adds an action listener for the corresponding button.

Class Controller

- `public class` **`Controller`**
- `extends java.lang.Object`

It has three attributes: two Polynomials objects p1 and p2 and a View object. It controls the whole system. It communicates data between the user to the application.

- ***Constructor Detail***

- **Controller**

```
public Controller()
```

The constructor Controller() is a public one, without parameters. Firstly, the constructor makes the frame

visible for the user. Then it implements the actions which should take place when a button is pressed (calling the methods from the View class). For each operation, the controller takes the list of coefficients from the text fields, converts the strings to Polynomials, compute the desired operation using the Polynomials objects and sets the text from the "labelResult" to the string representation of the output polynomial.

- ***Method Detail***

- **convertFromStringToPolynomial**

- `public pt2018.assign1.models.Polynomial convertFromStringToPolynomial(java.lang.String s)`
  `throws`
  `java.lang.NumberFormatException`

- Takes the string given as a parameter and converts it (if possible) into a Polynomial object.

Parameters:
`s - the string which represents a list of coefficients separated by a space one from the previous one`

Returns:
`a new Polynomial object`

Throws:
`java.lang.NumberFormatException - the method uses the parseInt() method which throws the exception`

5. Testing

What is tested	Input data	Output expected	Effective result	Pass/fail
Addition	1 0 -5 4 2 7 -9 5 3	8 -9 0 7 2	8 -9 0 7 2	Pass
Subtraction	1 0 -5 4 2 7 -9 5 3	-6 9 -10 1 2	-6 9 -10 1 2	Pass
Differentiation	1 0 -5 4 2	0 0 -10 12 8	0 0 -10 12 8	Pass
Integration	1 0 -5 4 2	1 0 -1.67 1 0.4	1 0 -1.(6)7 1 0.4	Pass
Multiplication	1 0 -5 4 2 7 -9 -9 3	7 -9 -44 76 23 -69 -6 6	7 -9 -44 76 23 -69 -6 6	Pass
Division	1 0 -5 4 2 7 -9 5	1.18 1.52 0.4	1.175(9)7 1.52 0.4	Pass

6. Conclusions

Future developments:

- the possibility to make operations with polynomials of higher order with higher coefficients
- a more convenient way to read the polynomials (it is inconvenient to place zeros for each missing term)
- compute and display the remainder of the division operation
- more efficient algorithms to perform the operations

- display the whole result of the polynomials with many terms
- improvement of the GUI

What I learn from the assignment:

- how to use javadoc
- how to use JUnit tests
- how to design a system using the MVC pattern