Fundamental Programming Techniques

Assignment 1:

Polynomial Calculator

Name: Filimon Adelin

Group: 30424

1.Objective

The objective of this assignment is to design a polynomial calculator with graphical user interface capable of basic operations.

The secondary objectives, which are mandatory for completing the principal one, are:

1. Design the classes for model (Monomial, Polynomial, etc.);
2. Test the model using Junit;
3. Design the graphical user interface;
4. Design the controller of the application;
5. Wire all the components together;
6. Test the application.

In the first step, all the components of the model are created making sure they respect the main objective. For that reason, there are more than 2 classes.

The next objective is to make sure the model is correctly designed and all the operations behaves as expected.

The third objective is to design the graphical user interface of the application using swing framework. In this step are created all the visual components (buttons, labels, etc.) and arranged in a panel using a layout manager. This module is the view of the application and receives data from the model to be displayed.

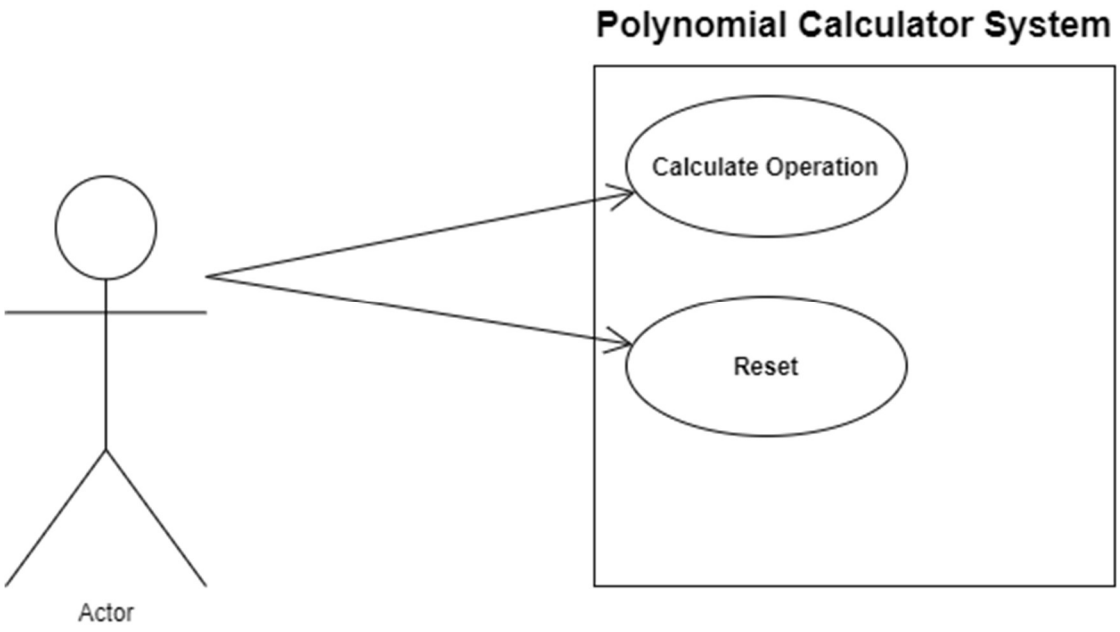
The next step is to design the controller which have the job to implement the application flow. Here we change the state of the calculator appropriately to the user requests. This module works with the other 2 components (view and model) in the same time.

By wiring the components the application is able to fully control all 3 modules (model, view, controller).

In the end, the entire project is tested to make sure that there are no holes in our design. Also, we make sure the program doesn't crush on certain inputs.

2.Problem analysis

The application must provide a polynomial calculator capable of computing different operations (SUM, DIFFERENCE, PRODUCT, DIVISION, INTEGRATION, DERIVATION) which means that the app must take the operands as input from the user, wait for an operation input and display the output. If the input provided is not valid the app must handle this situation with a pop-up message containing the reason of the problem.



Use Case: Calculate Operation

Primary Actor: User

Preconditions: -

Main success scenario:

1. The user provides a polynomial
2. The user provides an operation
3. The calculator checks if the inserted polynomial is valid
4. The calculator displays the first operand
5. The user provides the second operand, if it is the case
6. The user press “=” button
7. The calculator checks if the inserted polynomial is valid
8. The calculator displays the entire operation to be performed
9. The calculator displays the result of the operation

Alternative sequences:

a) Invalid first operand
   - The calculator throws a pop-up message specifying that the input is incorrect
   - The scenario returns to step 1

b) Invalid second operand
   - The calculator throws a pop-up message specifying that the input is incorrect
   - The scenario returns to step 5

Use case: Reset

Primary Actor: User

Preconditions: -

Main success scenario:

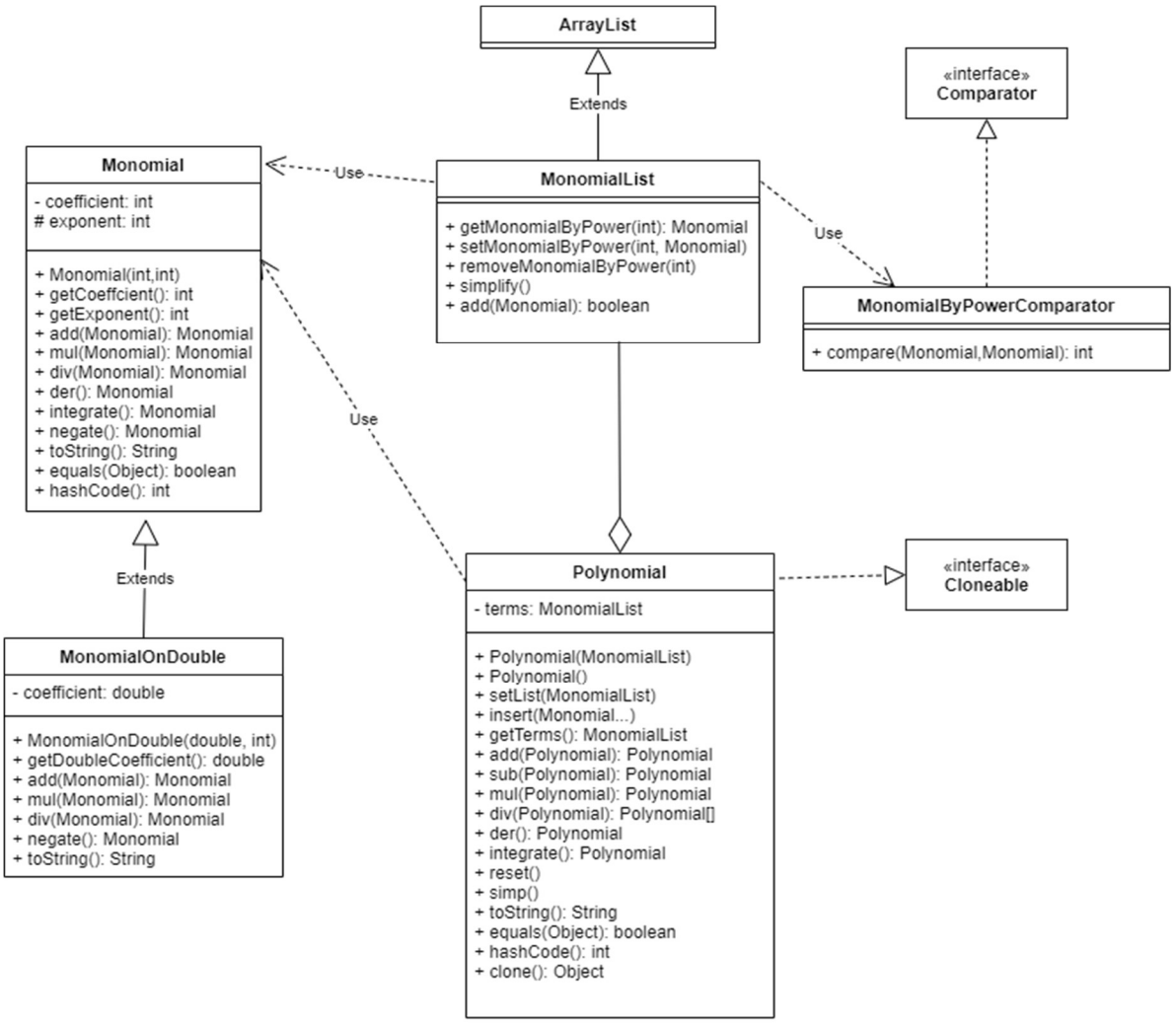
1. The user presses the reset button
2. The calculator is reset regardless the current state

3.Design of the application

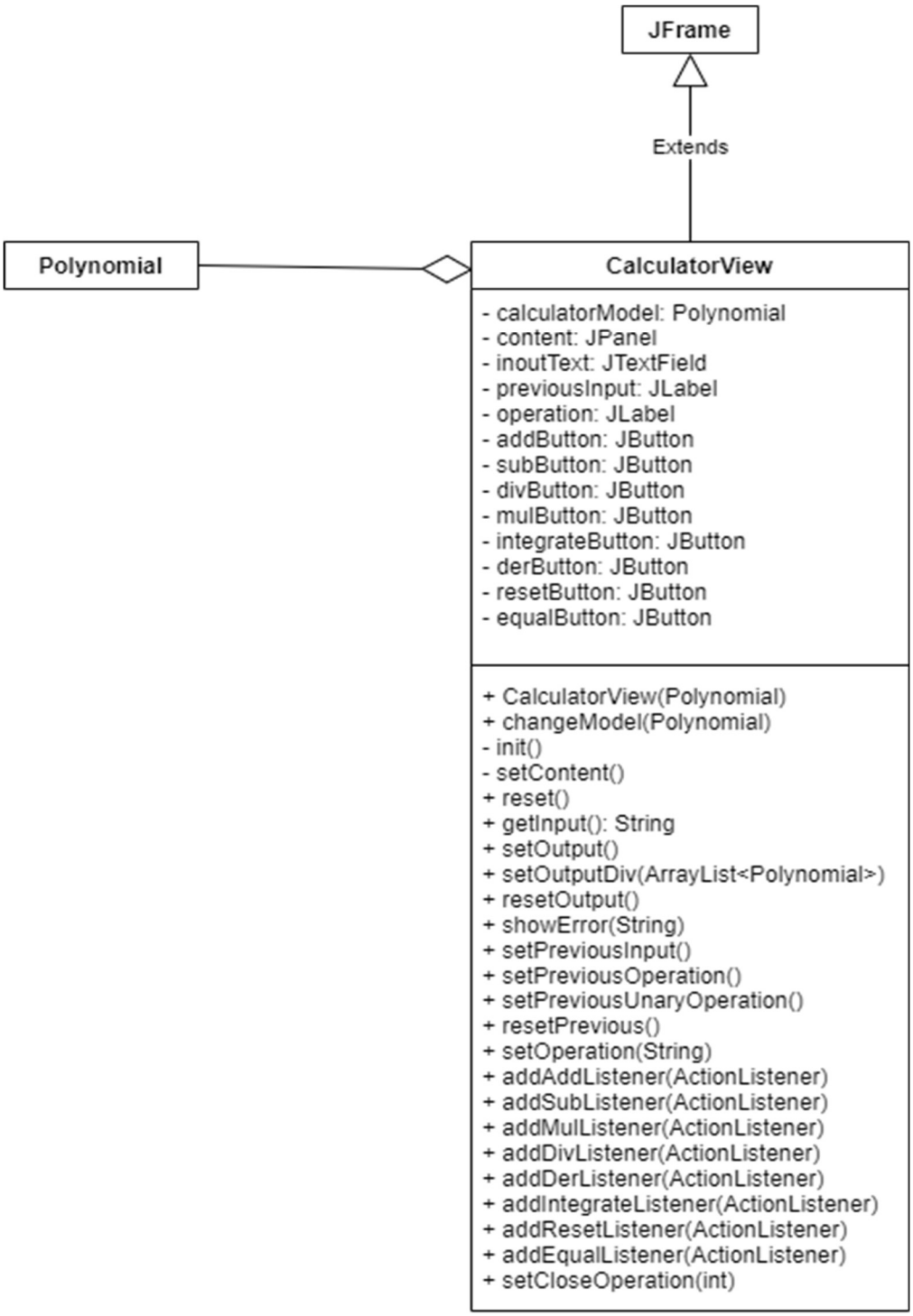
The application works only for monomials with integer coefficients, but at some operations, more specific, division and integration, the result is represented as a polynomial with double coefficient terms. For that reason, there exist 2 classes representing the monomials: Monomial, with integer coefficient, and MonomialOnDouble, with double coefficient. The last one will override the principal methods from Monomial in order to obtain the correct result from division and integration operations and also to print it appropriately.

Another important aspect of the design is the MonomialList class which is an ArrayList used only for Monomial terms. That class have special methods that query monomials by their exponent and another functionality is that of simplify method used for reducing the size of the list by combining the terms with the same exponent.

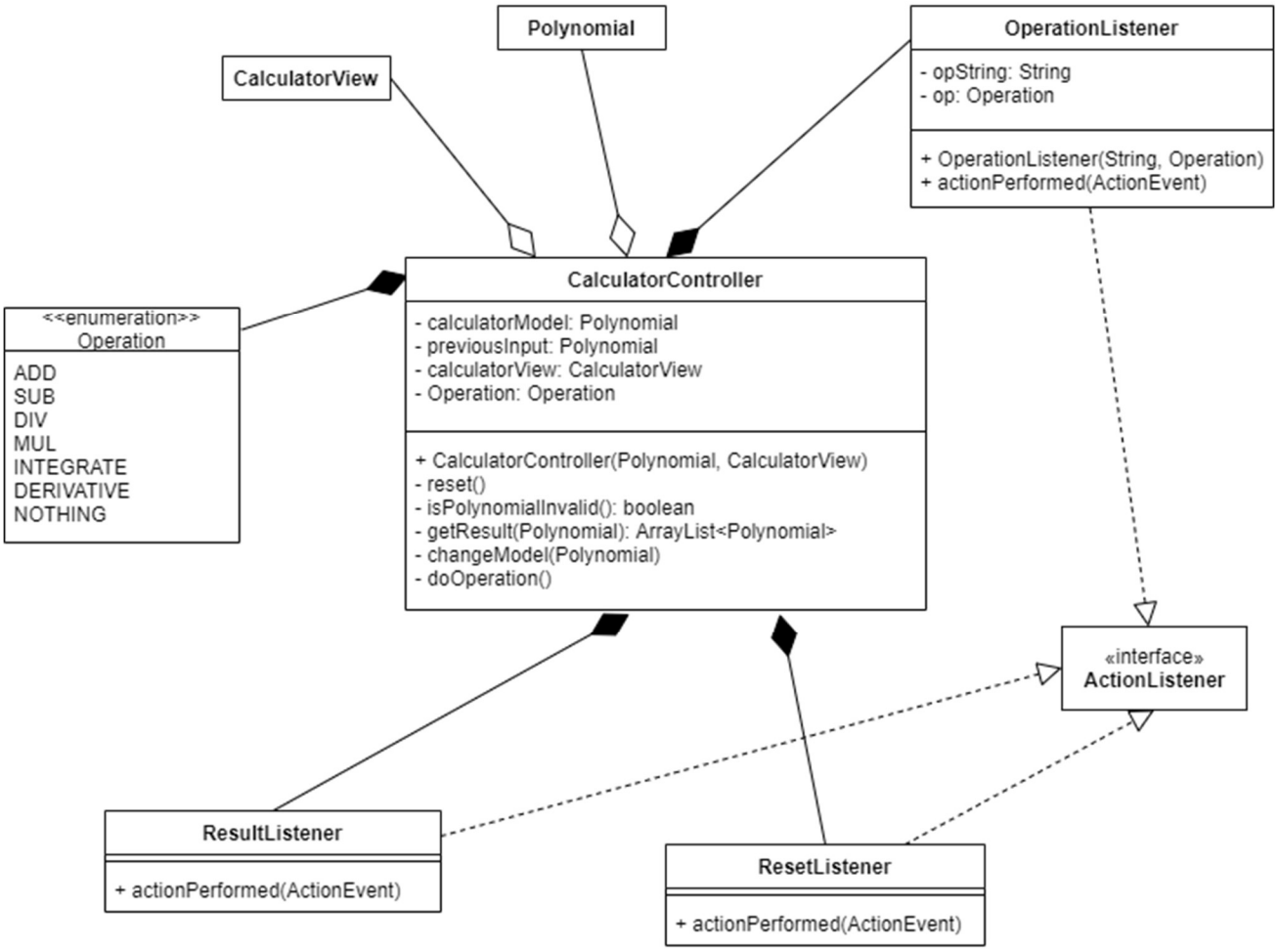
Next, it is presented the UML diagram of the model:



The graphical user interface of the application is described in a single class, in a new package, view:



The controller package is described as follows:



The application is designed in an MVC architectural pattern such that there are 3 packages: model, view and controller.

In model package there are all the logic behind the application like operations logic. The data structure used for maintaining the list of monomials is a defined class, MonomialList, which inherits from ArrayList and have some adjustments for finding or simplifying the terms. The algorithm used for the division of the polynomials is long division with pseudo-code[1]:

```
function n / d is
    require d ≠ 0
    q ← 0
    r ← n             // At each step n = d × q + r

    while r ≠ 0 and degree(r) ≥ degree(d) do
        t ← lead(r) / lead(d)       // Divide the leading terms
        q ← q + t
        r ← r − t × d

    return (q, r)
```

The user interface is designed using 2 dependencies: flatlaf[3], which provides the theme, and miglayout[2] which is a layout manager used for positioning the components.

4.Implementation

Next it is described the classes from model package:

Monomial class

That class describes a monomial with integer coefficient and integer exponent able to perform some basic operations.

Fields:

- coefficient: is private, it is initialized in constructor and have a getter
- exponent: is protected since it is used by MonomialOnDouble, it is initialized in constructor and have a getter.

Methods:

- Constructor: verify if exponent is less than 0 or coefficient equals 0, in that case it will throw an IllegalArgumentException.
- add (Polynomial): method which returns the sum between the caller and the given Polynomial as parameter. The method checks for null parameter, in which case it will return the caller since the null acts as a 0 monomial. Also, the method checks for exponent equality. The method will check if the parameter is an instance of MonomialOnDouble, in which case the double coefficient is stored and used lately. The resulting coefficient is obtained in double. Lastly, it is checked if the result is effectively a double, in which case it will return a MonomialOnDouble else it will return Monomial
- mull (Polynomial): returns the product between the caller and input parameter. Checks for null in which case it will return null. Like the add method it will check the actual class of the parameter and the return Monomial depends on the resulted coefficient.
- div (Polynomial): returns the quotient of the division between the caller and the input parameter. Checks for null and exponent issues in which case it will throw an exception. The return Monomial also depends on the resulted coefficient.
- der (): returns the derivative of the caller.
- integrate (): returns the integration of the caller. The result can be MonomialOnDouble if the coefficient is double

- negate (): returns a new Monomial with negated coefficient, used for subtraction of polynomials

MonomialOnDouble class

The class describes the monomials with double coefficient. It inherits the methods from Monomial and override some of them specific for division operation.

Fields:

- coefficient: is private double, and it is initialized in constructor and have a getter

Overrided methods:

- add (Monomial)
- mul (Monomial)
- div (Monomial)
- negate (Monomial)
- toString ()

MonomialList class

The class describes the list of monomials used by polynomial. The class inherits from ArrayList<Monomial>.

Methods:

- getMonomialByPower(int): returns the monomial with specified power. The search is in linear time.
- setMonomialByPower(int, Monomial): change the Monomial with specified power with a new one.
- removeMonomialByPower(int): removes the monomial with specified power.
- simplify (): merge the monomials with the same power.

MonomialByPowerComparator class

Comparator used for sorting the monomial list in descending order by their power. Implements Comparator interface.

Polynomial class

Describes a polynomial as a list of monomials. The class provides basic operations on polynomials.

Fields:

- terms: private list of monomials, type: MonomialList

Methods:

- Constructor (MonomialList): creates a polynomial based on the list received. The constructor also simplifies the terms.
- insert (Monomial): adds a monomial to the list
- setList (MonomialList): sets a new monomial list to the polynomial
- add (Polynomial): returns a new Polynomial as the sum between the caller and the input polynomial.

- sub (Polynomial): returns a new Polynomial as the difference between the caller and the input polynomial.
- mul (Polynomial): returns a new Polynomial as the product between the caller and the input polynomial.
- div (Polynomial): returns a Polynomial Array, the quotient and the remainder of the division between the caller and the input polynomial. The algorithm is presented at section 3.
- der (): returns a new Polynomial as the derivative of the caller.
- integrate (): returns a new Polynomial resulted by integration of the caller.
- reset (): clear the list of terms.
- simp (): simplify and sort the list in decreasing order of the power.
- clone (): returns a new polynomial with the same list as the caller.

The view package contains a single class, CalculatorView:

CalculatorView class

Fields:

- calculatorModel: the model of the application, of type Polynomial.
- content: the JPanel containing all the components of the application.
- inoutText: a textfield which acts as input and output in the same time.
- previousInput: label used for displaying the previous inputs.
- operation: label used for displaying the current operation.
- buttons for operations, reset button and equal button.

Methods:

- Constructor (Polynomial): defines the model, initialize the components and sets the content.
- changeModel (Polynomial): change the current model.
- init (): initialize the components and config the frame.
- setContent (): arrange the components using miglayout and sets the used font.
- reset (): reset the displayed text.
- Setters and getters for text components and setters of action listener for buttons.

The controller package:

CalculatorController class

Fields:

- calculatorModel: the model of type Polynomial.
- previousInput: Polynomial used for operations.
- calculatorView: the view of the application, of type CalculatorView.
- operation: enumeration of type Operation used for tracking the current state of the calculator.

Methods:

- Constructor (Polynomial, CalculatorView): sets the model and view and also sets the listeners of the buttons from view.
- reset (): method used for reset of model, view and operation.

- isPolynomialInvalid (): uses regex to parse the string from input text to polynomial, returns false if the parsing is not possible.
- getResult (Polynomial): takes the parameter the previous input and returns the result of current operation as an ArrayList of polynomials.
- changeModel (Polynomial): change the current model.
- doOperation (): executes the current operation.

Operation enumeration

Used for maintaining the track of the state of calculator.

Fields: ADD, SUB, DIV, MUL, INTEGRATE, DERIVATIVE, NOTHING

OperationListener class

Fields:

- opString: the string which will be parsed to operation
- operation: specify the current operation

Methods:

- actionPerformed (ActionEvent): change the view and the model of application by the user request.

ResultListener class

Methods:

- actionPerformed (ActionEvent): used for equal button, changes the model and view to the result of operation.

ResetListener class

Methods:

- actionPerformed (ActionEvent): used for reset button, reset the entire application.

5.Results

The framework used for testing is JUnit 4, next it is presented some test cases:

First class to be tested is Monomial since it is the principal player in our application. All the logic resides in this class and polynomial class just make use of that.

Firstly, there are not allowed monomials with negative exponents or zero coefficients. In that case our application should throw an IllegalArgumentException with a specific message.

The add function is checked for correct output, for that it is used assertEquals method:

assertEquals(new Monomial(-2**,**0)**,** t1.add(t5))**;**

That code will check if the t1 monomial (-4) added with t5 (2) will result in a new monomial with correct coefficient (-2).

There exist some special cases which needs to be checked such as:

- add with a null parameter should return a new monomial with the same coefficient and exponent as the caller since is an add with 0: *assertEquals*(t1**,**t1.add(null))**;**
- next it is verified if the sum between a monomial and a monomial with double coefficient will return a monomial depending on the result:
  assertEquals(new MonomialOnDouble(Math.*PI* + 7**,**3)**,** t4.add(t8))**;** where t4 is a monomial (7,3) and t8 is a monomial on double (3.14,3).
- the sum between a monomial and its negate should be null: *assertNull*(t1.add(t1.negate()))**;**

The mul function which should return the product between the caller and the parameter monomial is checked:

assertNull(t1.mul(null))**;** the multiplication with null returns null

assertEquals(new Monomial(-12**,**1)**,** t1.mul(t2))**;** where t1 has fields (-4,0) and t2 has fields (3,1)

The div function is checked for some exceptions which could appear:

```
try {
   t1.div(t8);
   fail ();
} catch (Exception ignored){}
```

The try block will throw an exception when t1 (-4,0) will try to divide t8 (3.14,3) since they have incompatible exponents

In the same manner is checked the rest of the model package.

The view and controller package don't have such test cases since they don't have logic which needs to be tested.

6.Conclusions

The application can be significantly improved. Some errors which should be corrected are

- MonomialList with its methods are not optimized, for example the usage of linear search is not so bad at small number of monomials but at big data it will cost time. The getMonomialByPower method will return the first encountered monomial with specified power. In this design there can be multiple monomials with the same power until the simplify method is called.

- The getPolynomial method is pretty ugly since it is full with if/else statements used for interpreting the data.

- The regex could be improved; the pattern will identify also the double coefficients but that is only for showing an error message.

- The changeModel method is not a good practice. The reference of model should never be changed. I used such a function because whenever I will do an operation, I will return a new monomial/polynomial with different reference and the model reference should be updated. A solution at that problem is to do operation on the caller itself without returning a new monomial/polynomial.

7.Bibliography

- Long division algorithm: https://en.wikipedia.org/wiki/Polynomial_long_division
- MigLayout: http://www.miglayout.com/
- FlatLaf: https://www.formdev.com/flatlaf/
- JUnit: https://www.vogella.com/tutorials/JUnit/article.html