|  |
| --- |
| Polynomial Calculator |
|  |
| March 156  Assignment 1  Nadu Laura-Andreea  Group 30421 |

Table of Contents

[Assignment Objective 3](#_Toc66714531)

[Main objective 3](#_Toc66714532)

[Sub-objectives 3](#_Toc66714533)

[Problem analysis, modeling, scenarios, use cases 3](#_Toc66714534)

[Analyzing the problem 3](#_Toc66714535)

[Modelling the solution 4](#_Toc66714536)

[Use case UML diagram 5](#_Toc66714537)

[Design Decisions 6](#_Toc66714538)

[Packages 6](#_Toc66714539)

[UML class diagram design 7](#_Toc66714540)

[Data Structures 7](#_Toc66714541)

[Algorithms 8](#_Toc66714542)

[Input Parsing 8](#_Toc66714543)

[Polynomial Addition 9](#_Toc66714544)

[Polynomial Subtraction 9](#_Toc66714545)

[Polynomial Multiplication 10](#_Toc66714546)

[Polynomial with a Monomial Multiplication 10](#_Toc66714547)

[Polynomial Division 10](#_Toc66714548)

[Polynomial Derivation 11](#_Toc66714549)

[Polynomial Integration 11](#_Toc66714550)

[Implementation 11](#_Toc66714551)

[Non-user interface classes 11](#_Toc66714552)

[Monomial Class: 11](#_Toc66714553)

[Polynomial Class: 11](#_Toc66714554)

[Operations Class: 12](#_Toc66714555)

[User interface classes 12](#_Toc66714556)

[Calculator Class: 12](#_Toc66714557)

[Controller Class: 13](#_Toc66714558)

[Results 13](#_Toc66714559)

[Conclusions 13](#_Toc66714560)

[Bibliography 14](#_Toc66714561)

# Assignment Objective

## Main objective

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials with integer coefficients, select the mathematical operation (i.e., addition, subtraction, multiplication, division, derivative, integration) to be performed and view the result.

## Sub-objectives

1. Analyze the problem and identify the requirements
2. Design the polynomial calculator
3. Implement the polynomial calculator
4. Test the polynomial calculator

# Problem analysis, modeling, scenarios, use cases

## Analyzing the problem

In mathematics, a polynomial is an expression composed of indeterminates (or variables) and coefficients (constants multiplied with the variables). Usually, this variable is denoted by “x” or “X”.

The general form of a polynomial in a single indeterminate X looks like this:

where are constants i {0, 1, 2, 3, …, n}. That is, a polynomial can either be zero or can be written as the sum of a finite number of non-zero terms. Each term consists of the product of a number – called the coefficient of the term – and a finite number of indeterminates, raised to nonnegative integer powers.

As it can be seen, a polynomial consists of a list of similar terms, called “monomials”. The monomials are the terms of the addition, meaning that they have the following general form where i {0, 1, 2, 3, …, n}. So for each monomial, we could retain only 2 numbers to reconstruct the general form. Those numbers could be expressed as a pair of (coefficient, exponent), i.e., where i {0, 1, 2, 3, …, n}.

Going back to the polynomial, we could write it as a set of monomial pairs.

This new form gives a hint about how to solve the problem.

The problem asks us to implement a polynomial calculator. A calculator should perform the basic operations like addition, subtraction, multiplication, division, derivation and integration. Thus, the system for polynomial processing receives as input one (for unary operations like derivation and integration) or two polynomials (for binary operations like addition and multiplication) with integer coefficients.

Functional requirements:

* The polynomial calculator should allow users to insert polynomials.
* The polynomial calculator should allow users to select the mathematical operation.
* The polynomial calculator should perform basic operations (i.e., addition, subtraction, multiplication, division, derivation, integration).
* The polynomial calculator should display the result of the operation performed.
* The polynomial calculator should display a specific method to select the desired operation.

Non-functional requirements:

* The polynomial calculator should be intuitive and easy to use by any user, regardless their knowledge.
* The polynomial calculator should verify the correctness of the input data inserted by the user.
* The polynomial calculator should simplify the input polynomials introduced by the user.
* The polynomial calculator should display a simplified version of the result.

## Modelling the solution

For a successful calculator, the user should be able to insert the desired polynomials, which represent the input data and to retrieve the result of one of the desired basic operations.

Operation

Calculator

LeftOperand

RightOperand

Result

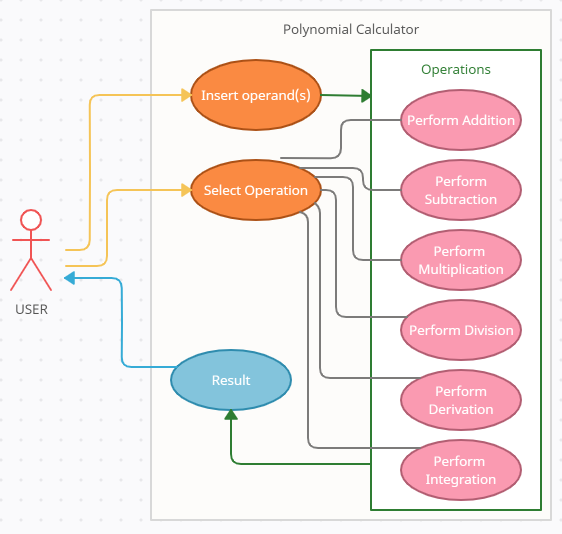
The “LeftOperand”, “RightOperand” and “Result” are polynomials. We would need a class called Polynomial to create the specific objects on which we should perform the operations. But just as said before, one polynomial could be represented as a set of monomials. Which means that the Polynomial object should actually be made of a set (or list) of Monomial objects.

Then there is the “Operation” which could be, as the problem specifies:

* + Addition of two polynomials with integer coefficients
  + Subtraction of two polynomials with integer coefficients
  + Multiplication of two polynomials with integer coefficients
  + Division of two polynomials with integer coefficients
  + Derivation of a polynomial with integer coefficients
  + Integration of a polynomial with integer coefficients

Both the insertion of the input data and the result of the operations would be processed by the User Interface which would contain input fields, operation buttons and output text (the result of the operation).

## Use case UML diagram

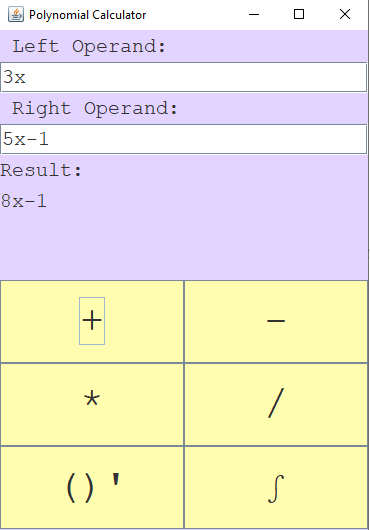
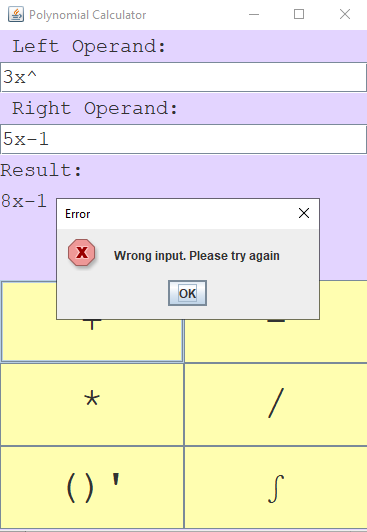
The use case diagram presented above has only one actor which in the case of the polynomial calculator is the user. This actor is the one that interacts directly with the calculator.

* Best Case Scenario

He/She/They can introduce one or two polynomials as operands and then choose an operation to perform from the list of possible operations. If the input data is introduced correctly, then the computer takes over and performs the selected operation, returning a result which the actor (user) can clearly see on the user interface.

* Worst Case Scenario

He/She/They should pay attention to the format of the input polynomials. Otherwise, if he enters a wrong polynomial format, then the calculator would display a “Wrong Input” error when selecting an operation and it will ask the user to re-introduce the input data.



# Spring MVC vs Angular. I've finished a few weeks ago a… | by Adriano Mota | Medium Design Decisions

## Packages

A clear design should be easy to understand by any new programmer that takes a look at the source code. Having that in mind, a good practice is to use the Model-View-Controller (MVC) architectural pattern.

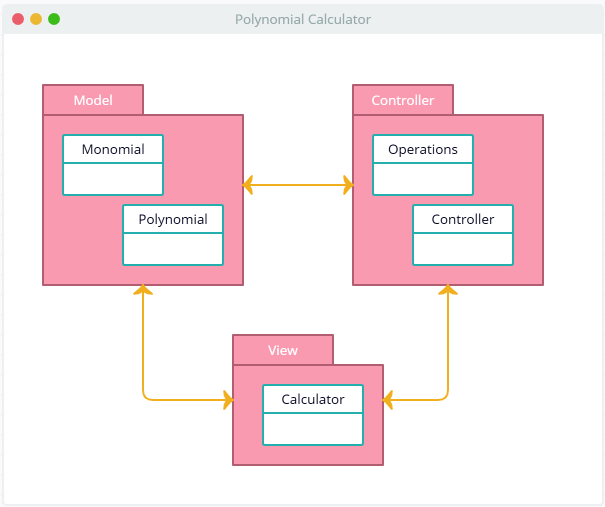
In the MVC architectural pattern there are 3 main packages:

1. Model is the central component. It mainly is the application’s data structure, independent of the user interface. Its scope is to manage the data directly, by the logic and the rules of the application, without the user interfering.

For the polynomial calculator, the Model contains 2 classes to manipulate data:

* 1. Monomial – as the name says, it is a class of an object that specifies the monomials of the polynomial.
  2. Polynomial – the main object of the problem. It is made of methods and a list of Monomials.

1. View is any graphical representation such as graphs, charts or the user interface. The calculator has in the View package the class Calculator which just defines the elements used to portray the user interface (buttons, fields and labels).
2. Controller is the class that accepts the input from the user and converts it as commands for the Models or Views. The package Controller contains 2 classes:
   1. Controller – this is the main controlling class that takes input from the user and transforms it into commands.
   2. Operations – this class is the operations class which takes the formatted input of the Controller and performs the basic polynomial operations with the Models (Polynomial and Monomial).



## UML class diagram design

The attributes and the methods of each class are presented in the “Implementation” section of the documentation where each class is particularly discussed in more detail.

## Data Structures

The data structures used in the application are mostly primitive data types, more specific I used int, double, boolean and String. These data types are used to store the coefficients and exponents of the monomials, to covert the fraction made by the numerator and denominator fields into a real value, to verify the correctness of the inputs and to parse the input from text form to polynomial for and vice versa.

Among these primitive types, a particular List named ArrayList was also used to store the monomials, instead of using the classic arrays. This choice was made because Lists have a more efficient performance than arrays regarding memory management (the size is not fixed and we could add as many variables as we want at the end of the said list) and access to their content.

## Algorithms

### Input Parsing

The user is introducing the input polynomials directly into a test field which allows him/her/them to submit anything they might think of. It is the programmer’s job to filter that input and see if there is anything to work with or if just the input text is too ambiguous and it can’t be used in real computation.

First, it’s important to check if the input was introduced correctly, which means:

* The only permitted characters are ‘x’, ‘X’, ‘-‘, ‘+’, ‘^’ and digits.
* No illegal combinations as:
* Double variable combinations like ‘xx’, ‘xX’, ‘Xx’ or ‘XX’.
* A power sign ‘^’ which is not preceded by a variable like ‘2^’.
* Negative exponents like ‘x^-2’.
* A power sign which is not followed by an exponent number.

This verification is done by the private method named validateInput() which is part of the Polynomial class. The method is called in the actual parsing method named createPolynomial() if the input is invalid, an exception is thrown.

On the other hand, if the input was correct, we may go on with the input parsing. First, we just remove all bank spaces so the input splitting could be done easier. After that, the actual parsing starts. We split the input into monomials using a Pattern and a Matcher with a monomial recognition regex:

Pattern pattern = Pattern.*compile*("((([\\+|-])?\\d\*)([x|X](\\^(\\d\*))?)?)");  
 Matcher matcher = pattern.matcher(input);

Then, for each monomial fitting the regular expression for monomials, we need to extract the coefficient and the exponent to create a pair just as explained in the “Analyzing the problem” section. To find these important numbers, from the matched input sequence we need to take a look at 3 important groups from the matcher:

* matcher.group(2) matches the coefficient with its sign. If there is no visible number found or any sign seen, then we can assume the coefficient is ‘1’. The same goes if we matched only a ‘-‘ sign, in which case the coefficient would be ‘-1’. If no one of these applies, then just parse the matched coefficient string to an integer type.

// if there was no visible number found or any sign, then we should just add a plus because  
 // the number is clearly positive and the coefficient is 1  
 if (matcher.group(2) == null || matcher.group(2).isEmpty() || matcher.group(2).equals("+"))  
 numerator = 1;  
  
 // if it has only a "-" sign, then the coefficient is -1  
 else if(matcher.group(2).equals("-"))  
 numerator = -1;  
  
 // if it is not 1 or -1, convert it to an Integer and add is as it is  
 else numerator = Integer.*parseInt*(matcher.group(2));

As it can be seen from the code, the coefficient is stored just into the numerator. That is because the input is guaranteed to have an integer coefficient, which means the denominator would be 1.

* matcher.group(4) matches a sequence that should contain the variable (if it is present in the monomial). If the variable is not present, then the monomial might have degree 1 or 0.
* matcher.group.(6) matches the corresponding monomial exponent. If there is no number found, then we migh have a degree 1 or 0 monomial so we would check group 4, if not, just parse it to an integer type.

### Polynomial Addition

// group 6 matches the exponent, while group 4 matches the variable (only x and X allowed)  
 // if nothing is found in group 6, there is no exponent visible => it may be 1 or 0  
 if (matcher.group(6) == null || matcher.group(6).isEmpty()) {  
 // if we can see an x (or X) then the exponent is 1  
 if(matcher.group(4)!=null && (matcher.group(4).contains("x")||matcher.group(4).contains("X")))  
 exponent = 1;  
 // if not, it is 0  
 else exponent = 0;  
 }  
 // if we can see it, we just add it  
 else exponent = Integer.*parseInt*(matcher.group(6));

In mathematics, polynomial addition is done by summing up the coefficients of the monomials which have the same degree.

The same rule was applied in the implemented algorithm. Taking into consideration that the operation would be applied only on the input polynomials which have integer coefficients, the denominator of each monomial was ignored because all of them would be equal with 1.

The algorithm goes just like a classic merging algorithm for 2 sorted arrays of different length. It follows the steps:

1. set 2 pointers (indexes) to the start of each polynomial involved in the addition.
2. As long as both indexes still point to a valid position inside the list.
   1. Get the corresponding monomials from each list.
   2. If their exponents differ, then add to the solution the one with the biggest exponent and increase the index of the corresponding polynomial.
   3. If they are equal, then create a new Monomial with the same exponent whose numerator would be the sum of the numerators of the input monomials. Add the new monomial to the solution and increase both indexes.
3. Upon reaching step 3, one of the indexes went past the size of the list. But we cannot know if the other one did the same. So:
   1. As long as the index of the first polynomial is still pointing inside the list, add the remaining monomials to the solution.
   2. As long as the index of the second polynomial is still pointing inside the list, add the remaining monomials to the solution.

### Polynomial Subtraction

Polynomial subtraction is done by subtracting the coefficients of the monomials which have the same degree. However, this time the subtraction is used also for division, which is why we must take into consideration that the operation would be applied only on polynomials which might have real coefficients. The denominator of each monomial must be taken into account.

The algorithm goes just like the addition algorithm, with some slightly changes. It follows the steps:

1. set 2 pointers (indexes) to the start of each polynomial involved in the addition.
2. As long as both indexes still point to a valid position inside the list.
3. Get the corresponding monomials from each list
4. If their exponents differ, then add to the solution the one with the biggest exponent and increase the index of the corresponding polynomial. If the one with the bigger exponent is associated to the right operand, then add the monomial tot the solution with an inverted sign for the coefficient. For example, -3x would become +3x just as +5x^2 would become -5x^2.
5. If they are equal, then create a new Monomial with the same exponent. The coefficient would be computed according to the mathematical rule:

where a, c = numerator of the left, respectively right, operand’s monomial, b = denominator of the left, respectively right, operand’s monomial.

1. Upon reaching step 3, one of the indexes went past the size of the list. But we cannot know if the other one did the same. So:
2. As long as the index of the first polynomial is still pointing inside the list, add the remaining monomials to the solution.
3. As long as the index of the second polynomial is still pointing inside the list, add the remaining monomials to the solution with an inverted sign.

### Polynomial Multiplication

Multiplication of two polynomials is done by multiplying every monomial from one polynomial to every monomial of the other polynomial. As for the addition, the polynomials used for multiplication are the ones inserted by the user, which are guaranteed to have integer coefficients, so the denominator may be ignored.

The multiplication process goes like this:

For each monomial in the left operand take all monomials in the right operand and multiply them. To multiply 2 monomials, add their exponents and multiply their numerators. Add the resulting monomial to the solution.

### Polynomial with a Monomial Multiplication

This operation is used as a helping method for the division algorithm. That is why, this operation could be applied to monomials with real coefficients.

For each monomial in the polynomial multiply them with the monomial by the rules:

* add their exponents.
* multiply their coefficients by the rule:
* add the resulting monomial to the solution.

### Polynomial Division

1. Set the remainder to the value of the left operand.
2. Divide the first term of the remainder by the highest term of the right operand (meaning the one with the highest power of x). The division of two monomials is done by subtracting the exponents and divide the real coefficients by the rule:
3. Multiply the right operand by the result just obtained (add it to the solution polynomial which would become a term of the eventual quotient).
4. Subtract the result of the multiplication from the remainder.
5. Repeat from step 2 until either the remainder is empty or its degree is smaller than the right’s operand degree.
6. Return the quotient and the remainder.

### Polynomial Derivation

Derivation is a unary operation. As per differentiating rules, the derivative of a sum of terms is the sum of the derivative of the terms.

The algorithm proposed traverses the terms of the polynomial, decreases the exponent of each one of them and multiplies the coefficient with the initial value of the exponent.

### Polynomial Integration

The integration of a polynomial is another unary operation which is done by the formula:

The algorithm proposed traverses the terms of the polynomial, increases the exponent of each one of them and divides the coefficient with the new value of the exponent.

# Implementation

## Non-user interface classes

### Monomial Class:

The role of this class is to define the structure of the terms of the monomials. Even though the problem specifies that the user may enter only polynomials with integer coefficients, after operations like division or integration those coefficients may have real values.

Because of the specifications of the problem, I couldn’t use the double type to store the coefficients. The solution I came up with is storing them as a separated fraction: one field for the numerator and one for the denominator. This way, the numerator and denominator may be integers.

Attributes of the class:

* exponent: an integer used to store the exponent of the corresponding term.
* numerator: an integer used to store the numerator composing the coefficient.
* denominator: an integer used to store the denominator composing the coefficient. For the case of integer coefficients, its value would be 1.

Constructors:

* Monomial (int exponent, int numerator): this method constructs a Monomial with an integer coefficient, the denominator being preset to 1.
* Monomial (int exponent, int numerator, int denominator): this method constructs a Monomial with a real coefficient, the denominator being sent as a parameter.

The only methods of the class are getters for all the fields.

### Polynomial Class:

This is the class that defines the whole polynomial.

Attributes of the class:

* polynomial: an array list that stores the terms of a polynomial.

Methods of the class:

* createPolynomial (String input): this is the parsing method whose algorithm is explained in the “Algorithms: section.
* printPolynomial(): this method traverses the polynomial array list and computes the corresponding string for each term. At the end, it returns the final printable string format of the polynomial.
* reducePolynomial(): this method takes in a polynomial, sorts it in the reverse order of the exponents of each term, proceeding to add the coefficients of the terms with equal exponents. At the end, the polynomial would be sorted and it would be in its most simplified form.
* validateInput (String input): a method to make sure that the user does not introduce illegal data formats.
* empty(): clear all the terms of a polynomial.

### Operations Class:

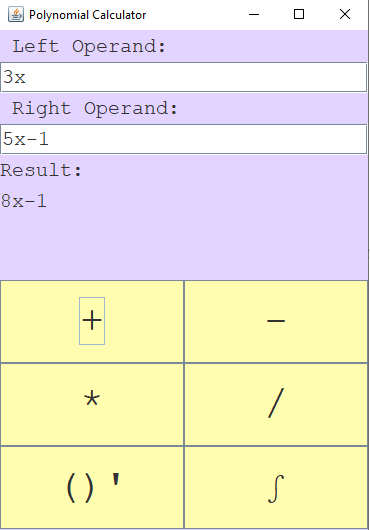
This is the class that computes all the operations applied on polynomials. All the methods of the class are static.

* addPolynomial(Polynomial P, Polynomial Q): returns the result of the addition of two polynomials based on the described algorithm.
* subPolynomial(Polynomial P, Polynomial Q): returns the result of the subtraction of two polynomials based on the described algorithm.
* multiplication(Polynomial P, Polynomial Q): returns the result of multiplying two polynomials based on the described algorithm.
* multiplication(Polynomial P, Monomial Q): returns the result of multiplying two polynomials based on the described algorithm.
* derivative(Polynomial P): returns the result of the derivation of a polynomial based on the described algorithm.
* integration(Polynomial P): returns the result of the integration of a polynomial based on the described algorithm.
* division(Polynomial P, Polynomial Q): returns the result of dividing two polynomials based on the described algorithm.

## User interface classes

### Calculator Class:

This is the Graphical User Interface class that defines all the graphical elements of the calculator. It was implemented using Java Swing so the attributes of the class consist of JButtons, JTextFields, JLabels, JPanels and a JFrame.



Frame

Field

Operations

The JFrame is the main frame of the application. This frame has a grid layout with 2 rows and 1 column. It contains 2 panels: a field panel and an operations panel.

The field panel also has a grid layout, but it has 7 rows and 1 column. This field will contain all the non-buttons elements. Here the user enters the polynomials and also here is the space where the results appear after the operation is selected.

The operations panel is another GridLayout panel with 3 rows and 2 columns. In this panel we can find the buttons corresponding to each possible operation. These buttons are the ones which the user presses in order to choose an operation.

These field are set up in the constructor of the class.

Beside the constructor, the only other methods are the getters.

### Controller Class:

This class contains just one method called “control”. Inside this method one can find the action listeners of the buttons. This class is the most important class of the application because it is the one that links the user and the front-end of the application to the models and the whole back-end application.

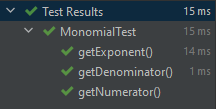
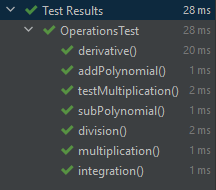
# Results

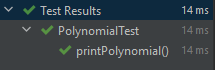
To make sure the back-end of the application works properly, I implemented tests using JUnit. The tests were implemented on the Operations, Monomial and Polynomial classes.

For each operation, monomial and polynomial methods that return a certain type, the asserEquals() method was invoked to make sure the results were the same as the expected.

For example, for the addition:

@Test  
 void addPolynomial() {  
 Polynomial leftOperand = new Polynomial();  
 Polynomial rightOperand = new Polynomial();  
  
 leftOperand.createPolynomial("x^2+x+1");  
 rightOperand.createPolynomial("-3x^4-3x+5x^2");  
  
 Polynomial test = Operations.*addPolynomial*(leftOperand, rightOperand);  
  
 *assertEquals*("-3x^4+6x^2-2x+1", test.printPolynomial());  
 }



Test results:

# Conclusions

As a conclusion for this homework, I can say that this assignment was a great way to keep the OOP concepts learned on the first semester and also to learn a few new things like how to use JUnit to test my code or just how to make an application using Java Swing.

For further improvements, I would implement the monomials using double from the start, because the use of fractions with numerators and denominators is really complicated and it make the code too hard to read when the solution could be easily achieved by using double coefficients.

I would also implement all of these operations to accept real input coefficients and, for the input parsing, I think I would rather use the String.split() method reather than pattern matching.

Finally, I would like to add features like computing the value of a polynomial for a certain value introduced by the user or just computing the roots of the polynomial.

# Bibliography

* Mathematical Documentations
  + [Polynomial Division, Wikipedia](https://en.wikipedia.org/wiki/Polynomial_long_division)
  + [Polynomial, Wikipedia](https://en.wikipedia.org/wiki/Polynomial#:~:text=In%20mathematics%2C%20a%20polynomial%20is,x2%20%E2%88%92%204x%20%2B%207.)
* Architectural Pattern
  + [Model-View-Controller, Wikipedia](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)
  + [MVC, TutorialsPoint](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)
  + [MVC Photo](https://medium.com/@motacsadriano/spring-mvc-vs-angular-c1bae44dea64)
* Regular Expressions
  + [Testing](https://regex101.com/)
  + [Java Regex – Matcher, Jakob Jenkov](http://tutorials.jenkov.com/java-regex/matcher.html)
  + [PHP Form Validation, CodeCademy](https://www.codecademy.com/learn/learn-php/modules/php-form-validation)
* Diagrams
  + [Drawing Tool, Creately](https://app.creately.com/diagram/jUycP6LwPCW/edit)
  + [UML Class Diagrams](http://pages.cs.wisc.edu/~hasti/cs302/examples/UMLdiagram.html)