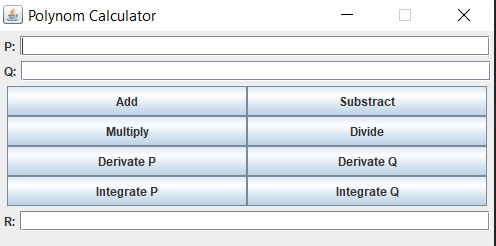
**Obiectiv principal**

Realizarea unui aplicatii de tip ‘calculator’ care poate efectua diverse operatii asupra polinoamelor prin simpla introducere a acestora si actionarea de diferite butoane in functie de operatie dorita.

Operatiile suportate vor fi:

* adunarea a doua polinoame;
* scaderea a doua polinoame;
* inmultirea a doua polinoame;
* impartirea a doua polinoame(reprezentata prin cat si rest);
* derivarea unui polinom;
* integrarea unui polinom.

Aplicatia interactioneaza cu utilizatorul prin intermediul unei interfete grafice simpliste prezentata in in figura urmatoare:



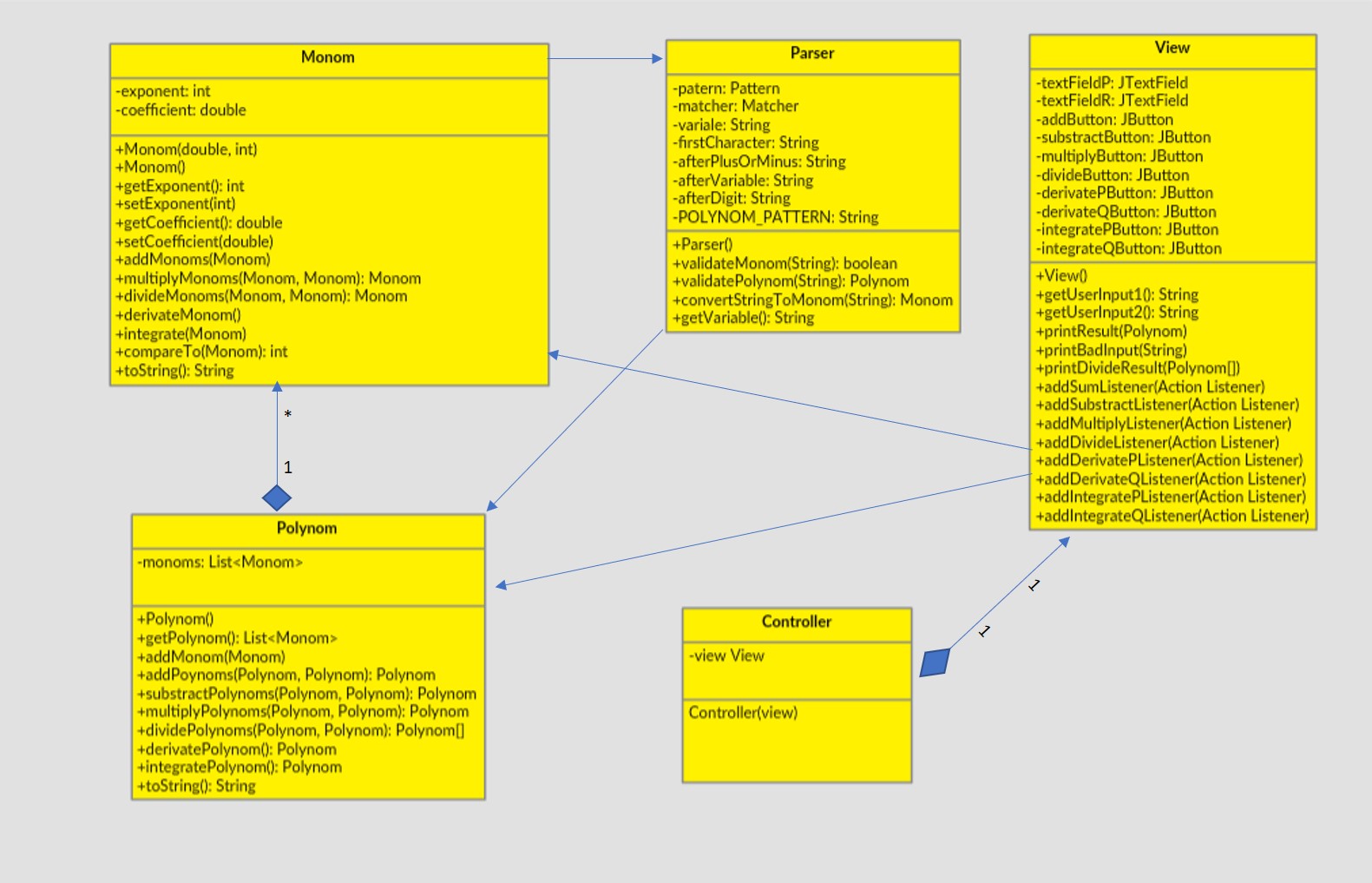
Interfata grafica constituita dintr-o fereastra ce contine:

* 2 campuri pentru introducere de text denumite, in mod generic, “P” si “Q” ;
* 8 butoane cu nume sugestive, cate unul pentru fiecare operatie implementata;
* un ultim camp de text denumit “R” unde se va face afisarea rezultatului.

Introducerea polinoamelor de catre utilizator se va face in campurile de text “P” si “Q”. Input-ul utilizatorului este apoi preluat din aceste campuri, verificat si interpretat astfel incat la apasarea unui buton pentru efectuarea operatiei dorite rezultatul operatiei sa apara imediat in campul destinat rezultatului (R).

**Implementare**

Aplicatia este implementata cu ajutorul limbajului de programare Java, drept urmare aceasta respecta paradigmele programarii orientate pe obiect.

In continuare avem diagrame de clase UML corespunzatoare:

Identificam cele 5 clase principale ale aplicatiei care pot fi grupate in categorii principale:

* prelucrarea datelolor: Parser;
* efectuarea operatiilor: Polynom si Monom;
* interfata grafica: View si Controller.

Diagrama UML prezinta 4 relatii de dependenta reprezentate prin sagetile normale si 2 relatii de compozitie care reprezinta faptul ca clasa Polynom nu poate exista fara clasa Monom respectiv clasa Controller nu poate exista fara clasa View.

**Descrierea claselor**

* **Monom**

La fel cum sugereaza si numele, reprezinta unul din elementele constitutive ale unui polinom. Clasa are 2 parametrii (coefficient si exponent) care reprezinta coeficientul respective exponentul unui monom.

Doi constructori, unul pentru a crea un polinom in functie de un coefficient si un exponent cunoscut si unul pentru a crea un monom initializat cu 0.

Metode getters&setters pentru a putea lucre cu variabilele din interiorul clasei.

Metoda addMonoms care efectueaza adunarea a doua monoame primate ca parametru si returneaza rezultatul sub forma unui monom nou.

Metoda multiplyMonoms care efectueaza operatia de inmultire a doua monoame.

Metodele divideMonoms, derivateMonom si integrateMonom care efectueaza operatiile de impartire, derivare respectiv integrare asupra monoamelor.

Metoda suprascrisa compareTo care ne permita ulterior sa comparam 2 monoame intre ele dupa putere.

Metoda suprascrisa toString care ne permite sa afisam un monom intr-un format text.

* **Polynom**

Clasa folosita pentru instantierea obiectelor de tip polynom. Contine un singur atribut, o lista de monoame ce reprezinta un polinom.

Metoda getPolynom ce ne permite sa obtinem lista de monoame din clasa Polynom.

Metoda addMonom care adauga un monom in lista de monoame din interiorul clasei.

Metodele addPolynoms, substractPolynoms, multiplyPolynoms, dividePolynoms care efectueaza operatiile de adunare, scadere, inmultire si impartire pe 2 polinoame primite ca parametru.

Metoda suprascrisa toString folosita pentru a ne ajuta sa afisam un polinom sub format text.

* **Parser**

Clasa care se ocupa cu verificarea input-ului primit de la utilizator pentru a ne asigura ca acesta poate fi reprezentat in program ca un polinom valid si mai apoi cu convertirea acestui input intr-un polinom.

Se foloseste java.util.regex pentru a imparti input-ul primit de la utilizator dupa un anumit pattern pentru a fi mai usor ulterior sa validam fiecare monom in parte fata de tot polinomul.

Variabilele pattern, matcher si POLYNOM\_PATTERN care se folosesc pentru a separa textul oferit de utilizator in posibile monoame.

String-urile firstCharacter, afterPlusOrMinus, afterVariable, afterDigit contin caracterele care pot urma dupa un anume character intr-un monom, folosite de asemenea pentru validare.

Variabila statica variable care ne asigura faptul ca ambele polinoame sunt de aceeasi variabila.

Metoda validateMonom care valideaza daca un string primit ca parametru poate are formatul valid al unui monom.

Metoda validatePolynom care se foloseste de metoda descrisa anterior pentru a valida fiecare monom ce alcatuieste polinomul verificand astfel daca string-ul introdus poate fi interpretat ca un polinom.

Metoda getVariable folosita ulterior pentru a obtine variabila corespunzatoare celor 2 polinoame introduse de utilizator.

* **View**

Clasa care se ocupa cu aspectul interfetei grafice si cu interactiunea dintre aceasta si utilizator.

Variabilele de tip JTextField: textFieldP, textFieldR, textFieldQ reprezinta cele 3 campuri de text “P”, “Q” si “R” unde utilizatorul poate introduca cele 2 polinoame asupra carora doreste sa efectueze o anumita operatie respective sa citeasca rezultatul operatiei dorite.

Variabilele de tip JButton: addButton, substractButton, multiplyButton, divideButton, derivatePButton, derivateQButton, integratePButton si integrateQButton care reprezinta cele 8 butoane prezente in interfata grafica, cate unul pentru fiecare operatie posibila.

Operatiile posibile sunt:

1. **Adunare**(prin apasarea butonului Add): aduna polinoamele introduse in campurile “P” si “Q” si afiseaza rezultatul in campul “R”.
2. **Scadere**(print apasarea butonului Substract): scade polinoamele introduse in campurile “P” si “Q” si afiseaza rezultatul in campul “R”.
3. **Inmultire**(prin apasarea butonului Multiply): inmulteste polinoamele introduse in campurile “P” si “Q” si afiseaza rezultatul in campul “R”.
4. **Impartire**(prin apasarea butonului Divide): imparte polinoamele introduse in campurile “P” si “Q” si afiseaza in campul “R” rezultatul sub forma: “<catul\_impartirii> | <restul\_impartirii>”.
5. **Derivare**(prin apasarea butonului Derivate P sau Derivate Q): deriveaza polinomoul introdus in campul “P” (Derivate P) sau “Q”(Derivate Q) si afiseaza rezultatul in campul “R”.
6. **Integrare**(prin apasarea butonului Integrate P sau Integrate Q): integreaza polinumul introdus in campul “P”(Integrate P) sau “Q”(Derivate Q) si afiseaza rezultatul in campul “R”.

Metodele getUserInput1 si getUserInput2 care sunt responsabile cu citirea textului introdus de catre utilizator in campurile “P” respectiv “Q”.

Metoda printResult care primeste ca parametru polinomul rezultat in urma efectuarii unei operatii si il afiseaza pe acesta in format text in campul destinat rezultatului: “R”.

Metoda printBadInput care afiseaza un mesaj de eroare in cazul in care input-ul primit de la utilizator este invalid. Mesajul de eroare este afisat tot in campul destinate rezultatului: “R”.

Metoda speciala printDivideResult care primeste ca parametru un Array de fix 2 polinoame reprezentand catul si restul impartirii a 2 polinoame si le afiseaza pe acestea in campul “R”, separate de separatorul “|”.

Metodele addSumListener, addSubstractListener, addMultiplyListener, addDivideListener, addDerivatePListener, addDerivateQListener, addIntegratePListener, addIntegrateQListener care fac corespondenta dintre apasarea unui buton si operatia care trebuie sa se efectueze odata cu aceasta.

* **Controller**

Clasa care face legatura dintre interfata grafica si clasele Polynom, Monom si Parser in vederea functionarii aplicatiei.

Variabila view instanta a clasei View care face legatura dintre Controller si intergata grafica.

Constructorul in care asignam fiecarui buton din interfata pasii pe care trebuie sa-I urmeze pentru a efectua operatia ceruta.

***Subclase ale clasei Controller:***

1. **sumButtonListener**:

- retine in variabilele userInput1 si userInput 2 cele 2 polinoame introduse de utilizator sub forma de text ;

-sterge toate spatiile din aceste variabile pentru a ajuta la validare ;

-apeleaza metoda validatePolynom din clasa Parser care returneaza un Polynom daca textul introdus de utilizator a putut fi convertit intr-un polinom valid altfel returneaza null;

- se verifica daca cele 2 polinoame obtinute sunt diferite de null(adica sunt valide) si se returneaza rezultatul operatiei de adunare prin apelarea metodei statice corespunzatoare din clasa Polynom sau se afiseaza mesajul “Bad input!” daca minim unul din cele doua polinoame introduse de utilizator sub forma de text nu a avut un format valid.

2. **substractButtonListener**:

-retine polinoamele introduse de utilizator sub froma de text in variabilele de tip String userInput1 si userInput2;

-se sterg toate spatiile din aceste variabile;

-apeleaza metoda validatePolynom pe ambele String-uri si retine valoarile returnate de aceasta in variabilele de tip Polynom p1 si p2;

-se verifica daca p1 si p2 sunt diferite de null(adica sunt valide) si se pune in campul destinate rezultatului valoarea obtinuta in urma operatiei de scadere efectuata de metoda statica substractPolynoms tot din clasa Polynom;

3.**multiplyButtonListener si divideButtonListener:**

-functioneaza la fel ca primele 2 subclase enumerate pana la pasul trei unde metoda apelata pentru obtinerea rezultatului este in aceste cazuri multiplyPolynoms respectiv dividePolynoms.

4.**derivatePButtonListener si derivateQButtonListener**

-se retine polinomul introdus de utilizator intr-unul din cele doua campuri (P pentru derivatePButtonListener si Q pentru derivateQListener) In variabila userInput;

-se elimina spatiile;

-se valideaza input-ul si se converteste din text intr-o variabila de tip Polynom(p);

-se afiseaza rezultatul in cazul in care valoarea din variabila p este diferita de null sau un mesaj de eroare in caz contrar.

5.**integrateQButtonListener si integrateQButtonListener**

-se retine polinomul introdus de utilizator intr-unul dintre cele doua campuri(P sau Q) in variabila de tip String userInput;

-se elimina spatiile din variabila userInput;

-se valideaza input-ul si se converteste din text intr-o variabila de tip Polynom;

-se afiseaza rezultatul stocat in variabila de tip Polynom daca aceasta este diferita de null sau mesajul “Bad Input!” in caz contrar.

**Mod de utilizare**

Utilizatorul deschide aplicatia si observa la inceput primele 2 campuri denumite sugestiv “P” si “Q” destinate introducerii celor 2 polinoame asupra carora urmeaza sa se faca validarea.

Pentru a simula ridica la putere se va introduce variabila urmata de caracterul “^” si tot ce se afla dupa aces caracter pana la urmatorul semn de adunare sau scadere va fi considerat puterea la care se afla respectiva variabila.

Atat coeficientii polinoamelor cat si puterile pot fi doar numere intregi, iar polinoamele trebuie sa fie de o singura variabila si ambele de aceeas variabila.

Restul descrierii se face in limbaj natural.

In continuare, utilizatorul poate sa apese pe unul dintre cele 8 butoane destinate fiecare unei operatii specifice.

Pentru butoanele de adunare, scadere, inmultire si impartire care se efectueaza cu ajutorul ambelor polinoame se valideaza ambele campuri “P” si “Q”. Daca cel putin unul dintre acestea nu trece de etapa de validare atunci se va afisa un mesaj de eroare.

Pentru butoanele Derivate P, Derivate Q, Integrate P, Integrate Q se valideaza doar campul corespunzator butonului, iar daca campul respectiv nu este valid se va afisa un mesaj de eroare.

Dupa etapa de validare, daca cele necesare au fost validate cu success, programul executa operatiile asupra polinoamelor sau asupra polinomului dorit(e) si apoi afiseaza rezultatul in campul “R” folosindu-se de cele doua metode suprascrise “toString” din calsele Monom si Polynom pentru a putea afisa Polynomul rezultat sub forma de text.

Daca etapa de validare nu a fost trecuta cu success atunci in campul final denumit generic “R” se va afisa mesajul de eroare “Bad Input!”.

**Parsarea**

Unul dintre procedeele cele mai complexe din spatele aplicatiei este “parsarea” unui string pentru a verifica daca acesta este un polinom valid. Parsarea reprezentata prin clasa Parser si metodele din aceasta este echivalentul etapelor de validare a unui string si conversie a string-ului respectiv descrise anterior.

Cu ajutorul java.regex si a pattern-ului “([+-]?[^-+]+)” impartim un posibil polinom in monoame in principal dupa semnele plus si minus.

Variabilele de clasa a caror denumire incepe cu “after” sunt declarate pentru a putea stii, in cadrul unui monom, pentru un caracter dat, care sunt caracterele valide care pot urma dupa acesta.

Luam fiecare monom obtinut in parte si prin intermediul functiei validateMonom si a celorlalte variabile de clasa vedem daca monomul sub forma text primit ca parametru respecta unul dintre formatele posibile ale unui monom. Pentru ca metoda validateMonom este de tip Boolean se va returna true daca monomul este valid si false in caz contrar.

Metoda validatePolynom este cea care se ocupa in principal de intreg procesul. Aceasta cheama metoda validateMonom pentru fiecare monom gasit, iar daca validarea are valoare true atunci polinomul introdus ca text poate fi si va fi convertit din text in formatul Polynom citindu-se din fiecare monom coeficientul si puterea, apelandu-se constructorul cu 2 parametrii din clasa Monm iar mai apoi adaugande-se fiecare monom in parte in polinomuil ce urmeaza a fi returnat. Inainte de returnare precum orice polinom este sortat descrescator dupa exponent pentru a face operatiile de adunare scadere, inmultire si impartire mai usoare. De asemenea aceasta metoda ne asigura si ca monoamele cu puteri egale vor fi “grupate” corespunzator astfel incat un polinom sa nu contina decat un singur monom la o anumita putere.

Convertirea unui monom din format text in format Monom se face prin intermediul metodei convertStringToMonom care parcurge caracter cu caracter string-ul primit ca parametru si identifica coeficientul si exponentul monomului urmand apoi sa creeze monomul ce urmeaza a fi returnat.