**DOCUMENTATIE**

Tema 1

Nume student: Mihali Gabriel

Grupa: 30225

**Cuprins**

[**1.** **Obiectivul temei** 3](#_Toc97663462)

[**2.** **Analiza problemei, modelare, scenarii, cazuri de utilizare** 4](#_Toc97663463)

[**3.** **Proiectare** 6](#_Toc97663464)

[**4.** **Implementare** 7](#_Toc97663465)

[**5.** **Rezultate** 10](#_Toc97663466)

[**6.** **Concluzii** 11](#_Toc97663467)

[**7.** **Bibliografie** 12](#_Toc97663468)

# **Obiectivul temei**

**Obiectivul principa**l al temei este implementarea unui calculator care face operatii pe polinoame.

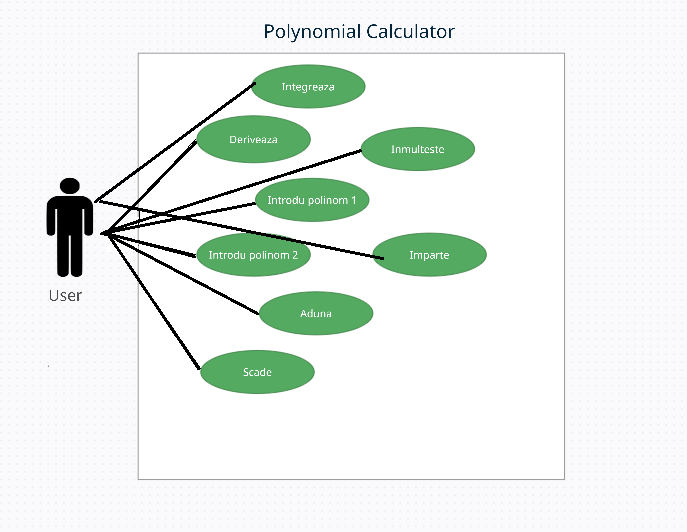
**Obiectivele secundare** sunt:

* **Implementarea unei clase Polinom :**
  + Aceasta clasa Polinom va avea in componenta sa mai multe Monoame. Mai multe despre clase si structura lor in capitolul 3 si 4.
* **Implementarea unei clase Monom :**
  + O clasa care va contine coeficientul si puterea. Mai multe despre clase si structura lor in capitolul 3 si 4.
* **Implementarea operatiilor(adunarea, scaderea, derivarea, integrarea, inmultirea, impartirea) :**
  + Mai multe in capitolul 4.
* **Tratarea cazurilor speciale :**
  + Inmultire cu 0, impartire cu 0, scaderea polinoamelor egale, derivarea unui termen liber, monoame cu coeficient 1, cu puterea 1, etc. Mai multe in capitolul 4.
* **Implementarea interfatei grafice :**
  + Aceasta va respecta arhitectura MVC si va fi intuitiva si usor de folosit. Mai multe despre interfata grafica in capitolul 4.
* **Testarea cazurilor speciale si a functionalitatii programului :**
* Se va face prin interfata grafica si prin JUnit. Mai multe despre acest obiectiv este descris in capitolul 4 si capitolul 5.
* **Respectarea arhitecturii MVC** :
  + Impartirea programului pe Controller, Model si View. Implementarea programului folosind arhitectura MVC este prezentata mai detaliat in capitolul 4.

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

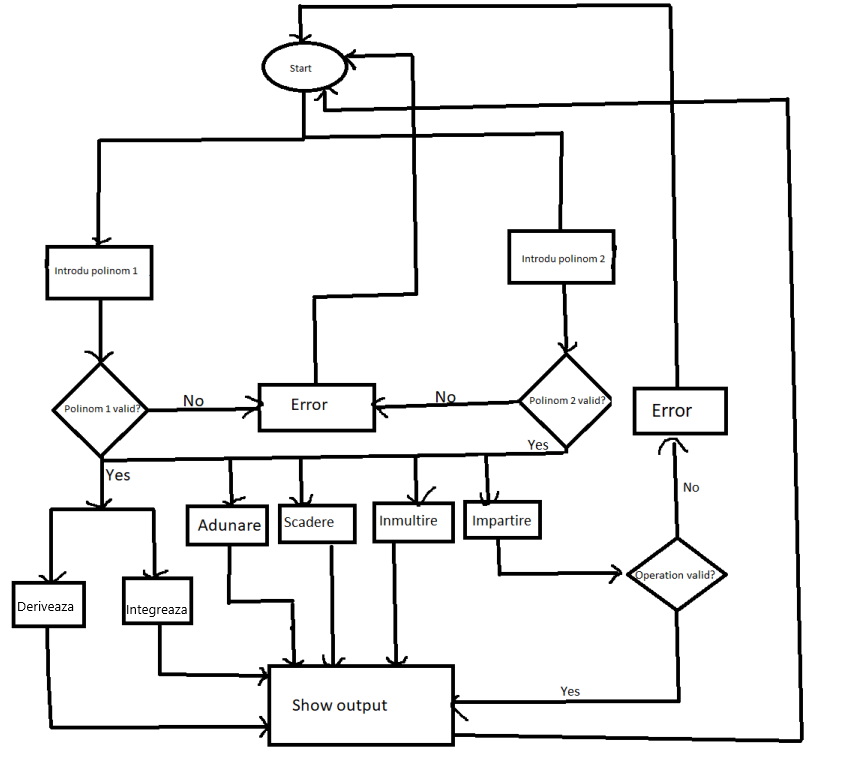
Implementati un calculator polinomial cu o interfata grafica prin care utilizatorul poate sa insereze polinoame, sa selecteze operatia matematica (adunare, scadere, inmultire, impartire, derivare, integrare) si sa I se afiseze rezultatul pe ecran.

* **Diagrama use-case-urilor:**

****

In imaginea de mai sus este prezentata diagrama use-case-urilor, adica operatiile pe care un utilizator le poate face in aplicatie.

* **Flow-Chart-ul use-case-urilor:**

****

In flow-chart-ul de mai sus este prezentat firul executiei aplicatiei, cu verificarile necesare la fiecare pas.

# **Proiectare**

* Aplicatia contine 9 clase cu o clasa HomeworkController, o clasa HomeworkView, si o clasa HomeworkModel.
* Respectand arhitectura MVC, clasa HomeworkView se ocupa de interfata grafica, clasa HomeworkController controleaza interfata si modelul, iar clasa HomeworkModel se ocupa de operatiile pe date.
* Acestea 3 sunt structurate in 3 package-uri diferite, in functie de rolul lor (package controller, package model si package view). Package-ul model contine cele mai multe clase: clasa HomeworkModel care contine operatiile de extragere polinom, de afisare polinom si operatiile aritmetice, 2 exceptii, IncorrectFormatException, care se ocupa de erorile de polinoamele introduse gresit si BadDivideException care se ocupa de impartirile gresite. De asemenea, package-ul model contine clasele Monom si Polinom. Clasa Monom contine coeficientul si puterea, iar clasa Polinom contine o lista de monoame.
* Avem si package-ul main, din care porneste aplicatia si contine o clasa Main cu un Controller, un Model si un View.

In imaginea de mai jos este prezentata diagrama UML a package-ului model:

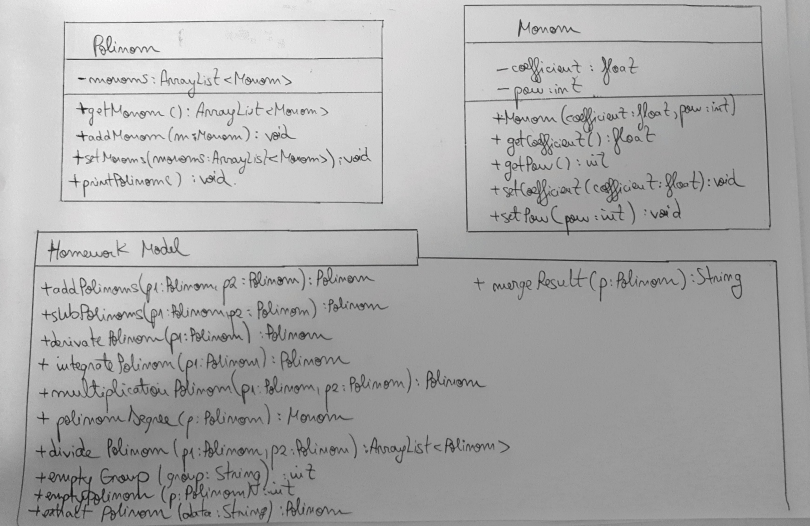


Diagrama UML a package-ului model descrie componenta claselor Polinom si Monom, cu campuri si metode si operatiile care se fac pe acestea in clasa HomeworkModel. De asemenea in clasa HomeworkModel avem si cateva metode de verificare, ca si emptyGroup, emptyPolinom si metode de formatare ca si extractPolinom sau mergeResult.

Diagrama UML a package-urilor:

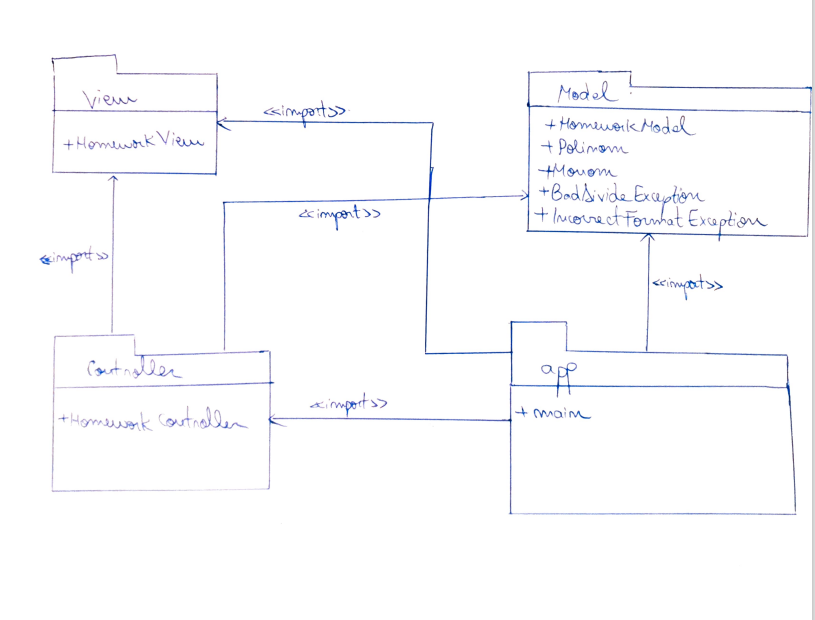


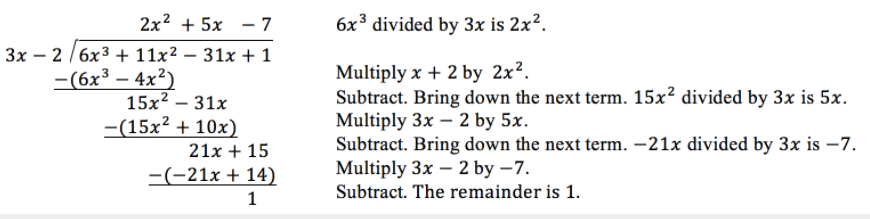
Diagrama UML a package-urilor descrie package-urile, clasele din componenta lor si cum sunt legate intre ele. Avem 3 package-uri, conform arhitecturii MVC, model, view si controller. Package-ul app este package-ul principal. Acesta importeaza toate cele 3 package-uri.

# **Implementare**

Aplicatia a fost implementata dupa modelul arhitecturii MVC. Contine 4 package-uri: app, model, view si controller.

* **Package-ul app**: Contine o clasa Main. Aceasta importa package-urile view, controller si model si are o metoda main in care se instantiaza 3 clase model view si controller cu care porneste aplicatia.
* **Package-ul model**: Contine 5 clase : HomeworkModel, Monom, Polinom si 2 exceptii , BadDivideException si IncorrectFormatException.
  + *Clasa HomeworkModel*: Este modelul propriu-zis si contine metodele pentru operatiile aritmetice pe polinoame, metode de verificare si metode care extrag polinoame din stringuri sau transforma polinoame in stringuri. Metoda addPolinoms primeste 2 polinoame p1 si p2 si returneaza suma celor 2 polinoame, intr-un nou polinom. Metoda subPolinoms returneaza diferenta dintre cele 2 polinoame. Metoda derivatePolinom returneaza derivata polinomului primit ca parametru. Datorita modelarii specifice a clasei Polinom, este necesara o verificare suplimentara asupra derivarii elementului 1, care rezulta 0. Metoda integratePolinom primeste ca parametru un Polinom si returneaza integrala polinomului. Metoda multiplicationPolinom, primeste ca parametru 2 polinoame si returneaza produsul dintre cele 2 polinoame. Aceasta metoda necesita verificari suplimentare, in cazul inmultirii cu 0. Metoda polinomDegree, primeste ca parametru un Polinom si returneaza Monomul cu cea mai mare putere din polinom. Aceasta metoda este necesara pentru algoritmul impartirii a 2 polinoame. Metoda dividePolinom primeste ca parametru 2 polinoame si returneaza rezultatul impartirii acestor 2 polinoame, un ArrayList de polinoame, care contine 2 elemente: catul si restul impartirii. Aceasta metoda arunca o exceptie de tip BadDivideException in cazul in care gradul polinomului 1, returnat de metoda polinomDegree, este mai mic decat gradul polinomului 2, in acest caz fiind o impartire subunitara. De asemenea se fac verificari pentru impartirea cu 0, care nu este permisa. Metoda impartirii a 2 polinoame dupa care este implementata aceasta metoda este ilustrat in imaginea de mai jos. Metoda extractPolinom primeste ca parametru un string, de obicei din interfata grafica, care trebuie modelat intr-un polinom si returneaza un Polinom. Aceasta foloseste o expresie de tip regex cu pattern-matching, ilustrata in imaginea de mai jos. Aceasta expresie regex desparte string-ul in 4 grupuri, dupa semn, variabila de tip x sau termen liber, coeficient si putere. Folosind aceste 4 grupuri, se gasesc monoamele polinomului si se formeaza polinomul. Sunt acoperite cazurile speciale ca si coeficient 0, puterea 1 si termenii liberi. De asemenea in aceasta metoda se verifica string-urile care nu se potrivesc format-ului de polinom, care au erori de scriere, ca si xx^2, 2^x, alte litere inafara de x. In cazul in care string-urile nu se potrivesc format-ului de polinom, metoda arunca exceptia IncorrectFormatException. Metoda mergeResult primeste ca parametru un String si returneaza un Polinom. Aceasta metoda pregateste un Polinom pentru a fi afisat in interfata grafica a utilizatorului. Aceata metoda se ocupa si de cazurile speciale, ca si coeficienti 1, puteri 1 sau 0.
  + *Clasa Monom:* Sta la baza operatiilor pe polinoame, este clasa care contine coeficientul si puterea, variabile de tip private. Astfel, avem setterele si getterele corespunzatoare.
  + *Clasa Polinom:* Aceasta clasa reprezinta polinomul in sine. Are ca si camp un ArrayList de monoame, de tip private, numit monoms cu settere si gettere corespunzatoare. Mai are metoda addMonom care primeste ca parametru un Monom si il adauga in ArrayList-ul monoms, pentru adaugarea pe parcurs. Mai are metoda setMonoms care primeste ca parametru un ArrayList de monoame si il pune in Polinom. Metoda printPolinom afiseaza Polinomul, adica lista de monoame.
  + *BadDivideException :* Este o clasa care extinde superclasa Exception si este aruncata de catre metoda dividePolinom in cazul impartirii gresite.
  + *IncorrectFormatException:* Este o clasa care extinde superclasa Exception si este aruncata de catre metoda extractPolinom. Aceasta exceptie se ocupa de erorile de formatare a input-ului utilizatorului.

Impartirea a 2 polinoame:

**

Expresie Regex folosita pentru a formata polinomul:



* **Package-ul view:** Contine o clasa, HomeworkView, care reprezinta interfata grafica a aplicatiei. Aceasta este implementata folosind Java Swing. Interfata grafica este un JFrame care continue mai multe JButton, pentru operatiile pe polinoame si 3 JTextField pentru input-ul utilizatorului si afisarea polinomului rezultat din operatii. De asemenea view-ul are 6 metode de addActionListener pentru fiecare JButton din interfata. Metoda getPolinom1 returneaza input-ul din primul TextField, adica primul polinom, iar metoda getPolinom2 returneaza input-ul din al doilea TextField, adica al doilea polinom. Metoda setResult, seteaza rezultatul operatiei dintre cele 2 polinoame in al 3-lea textfield. Metoda errorMessage deschide un JOptionPane de eroare, folosit pentru a afisa erori, iar metoda infoMessage deschide un JOptionPane de informatie, folosit pentru a afisa restul impartirii a 2 polinoame. In imaginea de mai jos este ilustrata interfata grafica.
* **Package-ul controller:** Contine clasa HomeworkController. Aceasta clasa este construita cu parametri de tip HomeworkModel si HomeworkView si utilizeaza aceste clase. Controller-ul utilizeaza clasa HomeworkModel pentru a face operatii pe polinoamele primite ca input din HomeworkView, si updateaza interfata, adica HomeworkView in functie de input si operatie. HomeworkController contine declaratii de clase de tip ActionListener: AdunareListener, ScadereListener, ImpartireListener, InmultireListener, IntegrareListener, DerivareListener. In aceste functii, intr-un try-catch, se apeleaza metode din model in functie de butonul apasat in interfata. In aceste clase ActionListener se fac niste verificari suplimentare pe String-urile primite ca input care mai apoi, vor fi trimise ca parametru in metoda extractPolinom.

Interfata grafica a aplicatiei:

Graphical user interface, application

Description automatically generated

# **Rezultate**

Verificarea operatiilor polinomiale a fost implementata cu JUnit. Clasa de test este HomeworkModelTest. Aceasta contine metodele addPolinoms, subPolinoms, derivatePolinom, integratePolinom, multiplicationPolinom si polinomDivide. Aceste metode testeaza operatiile de adunare, scadere, integrare, derivare, inmultire si impartire, folosind metoda assertEquals, care compara doua rezultate. Cele doua rezultate au fost obtinute astfel: primul rezultat a fost initializat in cod, rezultatul asteptat. Al doilea rezultat este rezultatul obtinut din operatia pe 2 polinoame, initializate, de asemenea, in cod. Acesta este rezultat obtinut. Daca cele 2 rezultate sunt identice, testul indica faptul ca operatia este implementata correct.

* Scenarii testare:

**addPolinoms:** polinom1 : 2x^3 + x^2 + 5x + 1

polinom2 : 5x^4 + 4x^2 + 2

rezultat : 5x^4 + 2x^3 + 5x^2 + 5x + 3

**subPolinoms:** polinom1 : 5x^4 + 3x^3 + x^2 + 1

polinom2 : x^5 + 5x^4 + 2x^3 + 2

rezultat : -x^5 + x^3 + x^2 – 1

**derivatePolinom:** polinom1 : 3x^6 + 15x^2 + 5

rezultat : 18x^5 + 30x

**integratePolinom:** polinom1: 10x^4 + 2x^3 + 15x^2 + 50

rezultat: 2x^5 + 0.5x^4 + 5x^3 + 50x

**multiplicationPolinom:** polinom1: 2x^3 + 2x + 2

polinom2: -x^5 + x^4 + 1

rezultat: -2x^8 + 2x^7 - 2x^6 + 2x^4 + 2x^3 + 2x + 2

**polinomDivide:** polinom1: 2x^3 – 3x^2 + 1

polinom2: x^2 – 2x

rezultat1: 2x + 1 (cat)

rezultat2: 2x + 1(rest)

# **Concluzii**

* In aceasta tema, s-au revizuit notiunile despre operatiile pe polinoame, s-au aprofundat cunostintele despre arhitectura MVC, despre cum trebuie structurata o aplicatie. Au fost asimilate cunostinte noi despre Maven, despre testarea unitara folosind JUnit.
* Aceasta aplicatie este foarte utila pentru utilizatorii care vor sa isi mareasca viteza de calcul. Este foarte versatila si utila in multe domenii. De asemenea, aceasta aplicatie poate fi extinsa, pot fi implementate operatii matematice mai complexe, ca si transformata Fourier, Laplace, sau alte operatii ce tin de domeniul ingineriei.

# **Bibliografie**

[**https://www.purplemath.com/modules/polydiv2.htm**](https://www.purplemath.com/modules/polydiv2.htm)

[**https://tugofweb.com/2021/01/02/polinoame-impartirea-polinoamelor-teorema-impartirii-cu- rest-teorema-restului-schema-lui-horner/**](https://tugofweb.com/2021/01/02/polinoame-impartirea-polinoamelor-teorema-impartirii-cu-%20%20rest-teorema-restului-schema-lui-horner/)

[**https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-running-unit-tests-with-maven/**](https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-running-unit-tests-with-maven/)

[**https://www.baeldung.com/java-new-custom-exception**](https://www.baeldung.com/java-new-custom-exception)

[**https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/**](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/)