MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

TEHNICI DE PROGRAMRE FUNDAMENTALE

TEMA 1

Calculator de polinoame

Dragoș-Bogdan Lazea, grupa 30223

An universitar: 2020 – 2021

1. Obiectivul temei

Obiectivul principal al acestei teme îl reprezintă proiectarea și implementarea unei aplicații care să funcționeze drept un calculator de polinoame, permițând efectuarea operațiilor de bază asupra polinoamelor și dispunând de o interfață grafică intuitivă pentru a facilita interacțiunea cu orice tip de utilizator.

Obiectivele secundare ale aceste teme sunt prezentate în tabelul de mai jos:

Obiectiv secundar	Descriere	Secțiune
1. Analiza problemei	- presupune analiza și delimitarea tuturor cerințelor pe care aplicația trebuie să le îndeplinească;	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
2. Identificarea scenariilor	- presupune identificarea și analiza tuturor scenariilor în care s-ar putea regăsi aplicația în funcție de valorile de intrare introduse de către utilizator (cazuri limită, excepții, funcționare corespunzătoare);	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Identificarea cazurilor de utilizare	- presupune identificarea situațiilor în care aplicația își justifică funcționalitatea;	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
4. Proiectarea propriu-zisă	- presupune parcurgerea etapelor de sintetizare și proiectare a aplicației conform paradigmei programării orientat pe obiect, prin realizarea de diagrame UML, proiectarea claselor, imaginarea interfeței grafice cu utilizatorul etc.	3. Proiectare
5. Implementarea aplicației	- presupune dezvoltarea propriu zisă a claselor și metodelor aferente acestora în cod scris în limbajul de programare Java;	4. Implementare
6. Testarea aplicației implementate	- presupune testarea corectitudinii rezultatelor furnizate de aplicația implementată în diverse scenarii de utilizare, folosind JUnit;	5. Rezultate

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

a. Analiza problemei

Problema de soluționat este reprezentată de faptul că efectuarea manuală a operațiilor cu polinoame este costisitoare și greoaie.

Prin urmare, a fost identificat, în urma analize problemei menționate, următorul cadru de cerințe funcționale:

- Aplicația trebuie să poată permite introducerea a două polinoame asupra cărora să se efectueze operațiile. De menționat este faptul că în cazul în care operația necesită un singur polinom (operațiile de derivare și integrare), primul polinom introdus va fi cel asupra căruia se va efectua operația, astfel nefiind necesară introducerea și celui de-al doilea polinom;
- Aplicația trebuie să informeze utilizatorul dacă inputul introdus este invalid;
- Aplicația trebuie să poată permite selectarea operației ce se va efectua asupra polinoamelor introduse;
- Aplicația trebuie să permită efectuarea următoarelor operații matematice asupra polinoamelor: adunare, scădere, înmulțire, împărțire, derivare și integrare.
- Aplicația trebuie să informeze utilizatorul referitor la eventualele erori și excepții matematice care pot apărea pe parcursul efectuării operațiilor, spre exemplu, împărțirea la 0.
- Aplicația trebuie furnizeze rezultatul corect al operației selectate;
- Aplicația trebuie să permită ștergerea inputurilor introduse și a rezultatului în vederea pregătirii pentru efectuarea unei eventuale noi operații;

Cadrul cerințelor nefuncționale conturate în jurul problemei de rezolvat este:

- Aplicația trebuie să dispună de o interfață grafică intuitivă.
- Aplicația trebuie să fie ușor de utilizat pentru orice tip de utilizator, indiferent de domeniul de activitate al acestuia.

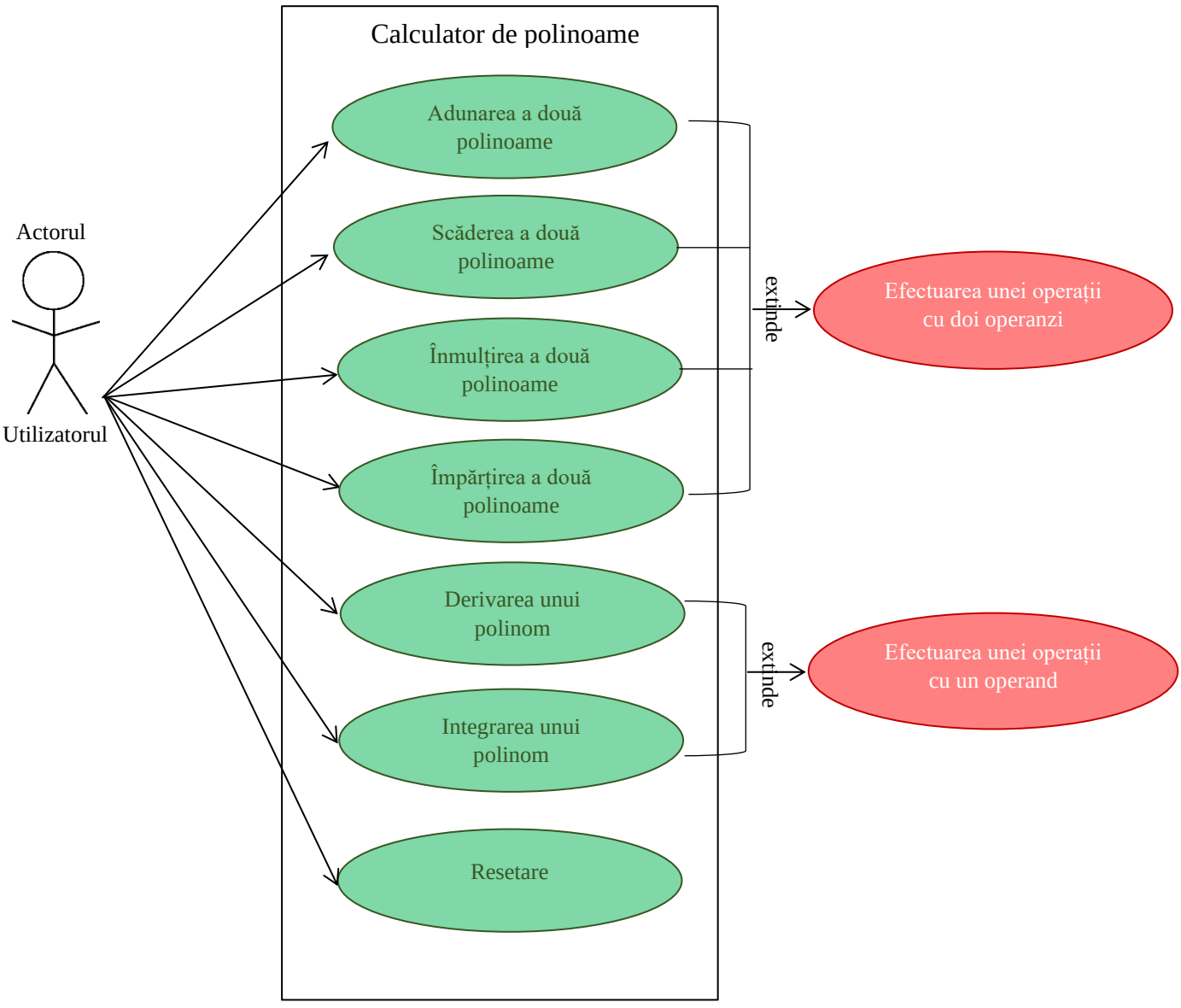
b. Modelare

Calculatorul a fost proiectat pe modelul unui sistem cu patru intrări și o ieșire, intrările fiind reprezentate de cele două polinoame asupra cărora se efectuează operația, o intrare de selecție a operației și o intrare care să permită resetarea calculatorului în vederea efectuării unui nou calcul, în timp ce ieșirea este reprezentată de polinomul rezultat în urma operației.

Modelul matematic al problemei este apoi translatat într-un model orientat pe obiecte, ce are la bază clasele Monomial, Polynomial și Operations prin intermediul cărora se vor defini și transpune în modelul obiectual intrările și ieșirile aplicației, definite pe baza modelului matematic.

c. Cazuri de utilizare și scenarii posibile

Diagrama cazurilor de utilizare este prezentată mai jos:



În cele ce urmează vor fi descrise cazurile de utilizare alte aplicației dezvoltate:

Use Case: Adunarea a două polinoame
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce două polinoame în interfața grafică a aplicației.
2. Utilizatorul selectează operația de adunare prin efectuarea unui click pe butonul „+”.
3. Calculatorul de polinoame execută operația de adunare a celor două polinoame și furnizează prin intermediul interfeței grafice rezultatul acesteia.

Scenarii alternative (posibile erori): Cel puțin unul dintre polinoamele introduse este invalid (nu respectă forma generală $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$, unde a_i, $i = 1, n$ sunt numere întregi)

1. Utilizatorul este notificat prin apariția unei ferestre cu mesajul de eroare „ Error: Invalid polynomial!”.
2. Scenariul se întoarce la pasul de introducere a polinoamelor de către utilizator.

Use Case: Scăderea a două polinoame
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce două polinoame în interfața grafică a aplicației.
2. Utilizatorul selectează operația de adunare prin efectuarea unui click pe butonul „-”.
3. Calculatorul de polinoame execută operația de scădere a celor două polinoame și furnizează prin intermediul interfeței grafice rezultatul acesteia. De menționat este faptul că primul polinom este considerat ca fiind descăzutul, iar cel de-al doilea este scăzătorul.

Scenarii alternative (posibile erori): Cel puțin unul dintre polinoamele introduse este invalid (nu respectă forma generală $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$, unde a_i, $i = 1, n$ sunt numere întregi)

1. Utilizatorul este notificat prin apariția unei ferestre cu mesajul de eroare „ Error: Invalid polynomial!”.
2. Scenariul se întoarce la pasul de introducere a polinoamelor de către utilizator.

Use Case: Înmulțirea a două polinoame
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce două polinoame în interfața grafică a aplicației.
2. Utilizatorul selectează operația de adunare prin efectuarea unui click pe butonul „*”.
3. Calculatorul de polinoame execută operația de înmulțire a celor două polinoame și furnizează prin intermediul interfeței grafice rezultatul acesteia.

Scenarii alternative (posibile erori): Cel puțin unul dintre polinoamele introduse este invalid (nu respectă forma generală $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$, unde a_i, $i = 1, n$ sunt numere întregi)

1. Utilizatorul este notificat prin apariția unei ferestre cu mesajul de eroare „Error: Invalid polynomial!”.
2. Scenariul se întoarce la pasul de introducere a polinoamelor de către utilizator.

Use Case: Adunarea a două polinoame
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce două polinoame în interfața grafică a aplicației.
2. Utilizatorul selectează operația de adunare prin efectuarea unui click pe butonul „/”.
3. Calculatorul de polinoame execută operația de împărțire a celor două polinoame și furnizează prin intermediul interfeței grafice rezultatul acesteia. De menționat este faptul că primul polinom este considerat ca fiind deîmpărțitul, iar cel de-al doilea ca fiind împărțitorul.

Scenarii alternative (posibile erori):

- Cel puțin unul dintre polinoamele introduse este invalid (nu respectă forma generală $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$, unde a_i, $i = 1, n$ sunt numere întregi)
    1. Utilizatorul este notificat prin apariția unei ferestre cu mesajul de eroare „Error: Invalid polynomial!".
    2. Scenariul se întoarce la pasul de introducere a polinoamelor de către utilizator.

- Cel de-al doilea polinom introdus este polinomul nul (0), caz în care se încearcă împărțirea la 0
    1. Utilizatorul este notificat prin apariția unei ferestre cu mesajul de eroare „Error: Division by 0!".
    2. Scenariul se întoarce la pasul de introducere a polinoamelor de către utilizator.

Use Case: Derivarea unui polinom
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce primul polinom în interfața grafică a aplicației.
2. Utilizatorul selectează operația de adunare prin efectuarea unui click pe butonul „d/dX".
3. Calculatorul de polinoame execută operația de derivare a primului polinom și furnizează prin intermediul interfeței grafice rezultatul acesteia.

Scenarii alternative (posibile erori): Primul polinom introdus este invalid (nu respectă forma generală $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$, unde a_i, $i = 1, n$ sunt numere întregi)

1. Utilizatorul este notificat prin apariția unei ferestre cu mesajul de eroare „Error: Invalid polynomial!".
2. Scenariul se întoarce la pasul de introducere a polinoamelor de către utilizator.

Use Case: Integrarea unui polinom
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce primul polinom în interfața grafică a aplicației.
2. Utilizatorul selectează operația de adunare prin efectuarea unui click pe butonul „ $\int$ ".
3. Calculatorul de polinoame execută operația de integrare a primului polinom și furnizează prin intermediul interfeței grafice rezultatul acesteia.

Scenarii alternative (posibile erori): Primul polinom introdus este invalid (nu respectă forma generală $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$, unde a_i, $i = 1, n$ sunt numere întregi)

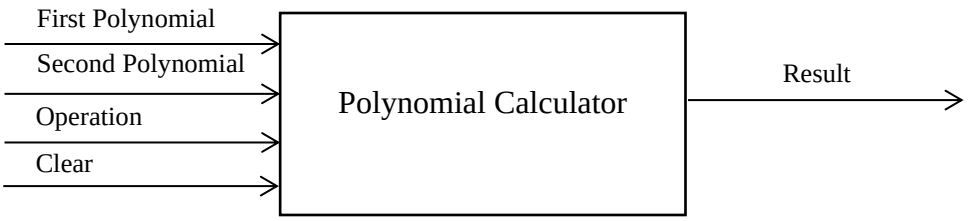
1. Utilizatorul este notificat prin apariția unei ferestre cu mesajul de eroare „Error: Invalid polynomial!".
2. Scenariul se întoarce la pasul de introducere a polinoamelor de către utilizator.

Use Case: Resetarea
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul efectuează click pe butonul „Clear all".
2. Calculatorul de polinoame se resetează prin ștergerea din interfața grafică a polinoamelor introduse anterior și a rezultatelor afișate, fiind pregătit în vederea efectuării unei eventuale noi operații cerute de către utilizator.

3. Proiectare

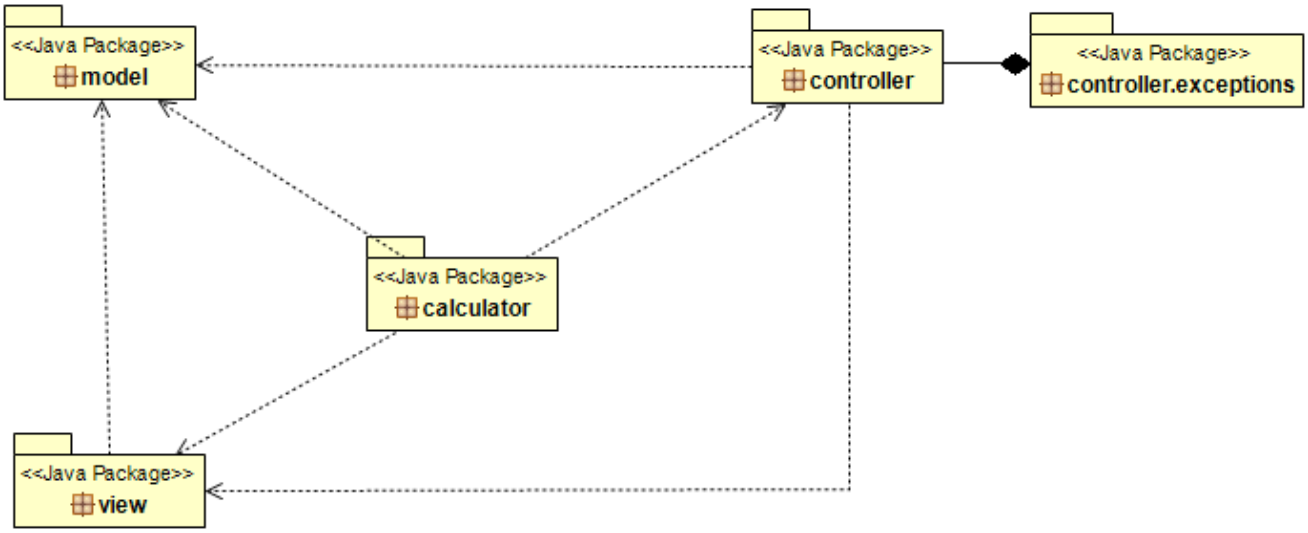
Primul nivel de proiectare îl reprezintă imaginea de ansamblu asupra aplicației prin realizarea schemei bloc a aplicației (cutia neagră a sistemului), care poate fi vizualizată mai jos.



Următoarea etapă a proiectării este reprezentată de divizarea aplicației în subaplicații cu funcționalități specializate. Subsistemele componente ale aplicației fiind organizate conform pattern-ului arhitectural MVC (**Model-View-Controller**), următorul nivel de divizare al aplicației putând fi identificat în diagrama de pachete prezentată mai jos.

Modelul încapsulează datele specifice aplicației și definește obiectele, logica și computațiile care stochează, manipulează și procesează datele furnizate de utilizator, în timp ce vederea redă conținutul modelului, fiind responsabilă de realizarea interfeței grafice cu utilizatorul, iar controllerul traduce interacțiunile utilizatorului cu vederea în acțiuni pe care modelul le poate executa.

Astfel, calculatorul de polinoame va fi compus din cele trei componente menționate mai sus, interconectate astfel încât să formeze un ansamblu unitar.

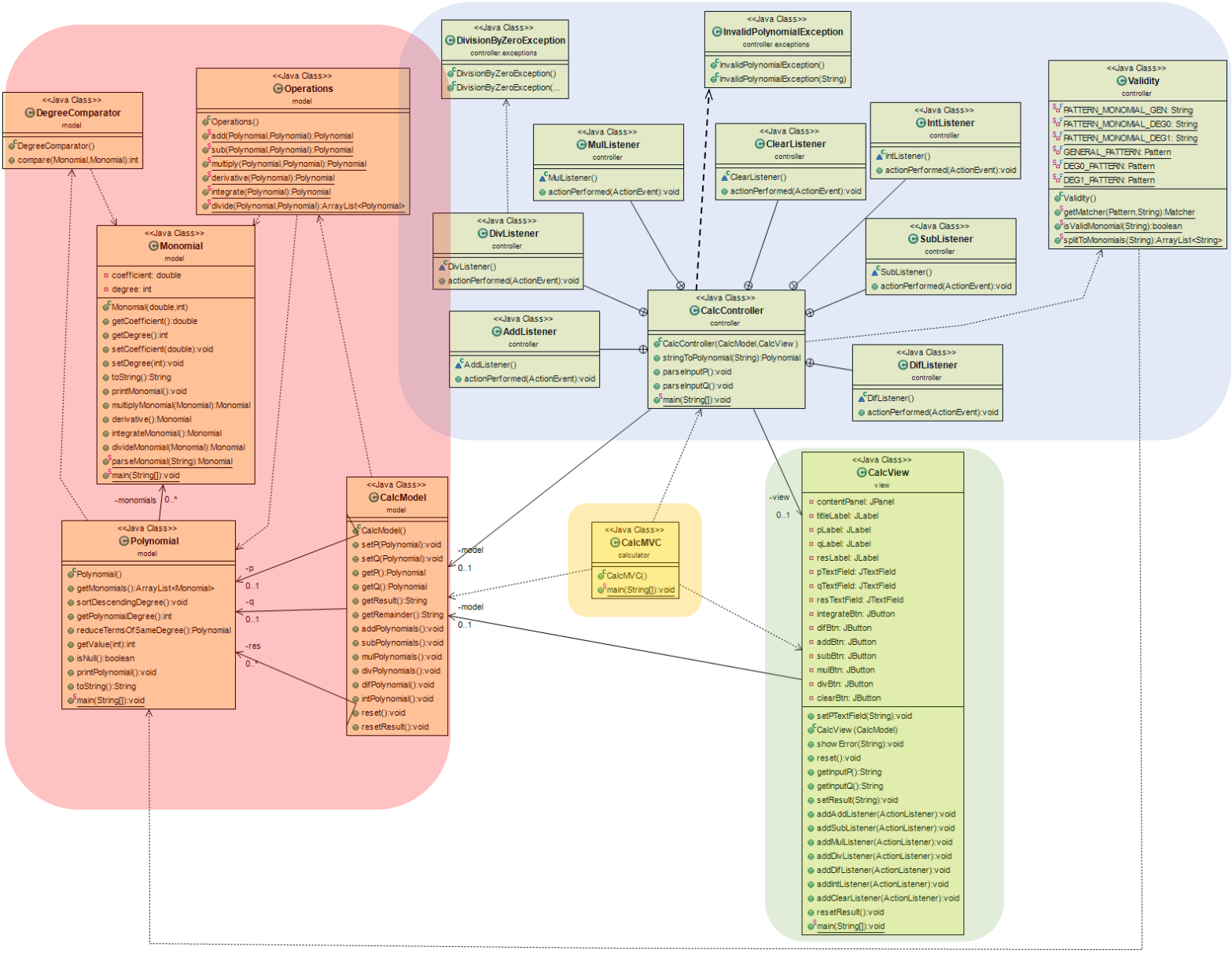


Trecând la cea de-a treia etapă de proiectare, se continuă descompunerea problemei și astfel se definesc cele mai de jos unități din descompunerea aplicației conform paradigmei programării orientate pe obiecte (POO), și anume clasele definite de dezvoltator.

Astfel, în cadrul modelului, responsabil de efectuarea computațiilor necesare îndeplinirii cerințelor funcționale, se dovedesc a fi necesare clase pentru definirea unor obiecte precum Monomial, Polynomial, reprezentat ca o listă de monoame, clasa Operations fiind responsabilă de implementarea operațiilor ce pot fi efectuate folosind aplicația, în timp ce clasa CalcView este responsabilă de realizarea interfeței grafice.

Descrierea tuturor claselor va fi detaliată în secțiunea dedicată implementării aplicației, împreună cu metodele importante și definitorii fiecărei clase din componența proiectului.

În cele ce urmează va fi expusă diagrama UML de clase ale aplicației, denumirile acestora fiind sugestive și inspirate din domeniul problemei. Colorarea diferită a grupurilor de clase din diagrama de mai jos indică apartenența acestora la pachetele expuse în diagrama de pachete, prezentată anterior.



Dintre algoritmii utilizați la proiectarea soluției, relevant de analizat în contextul dezvoltării aplicației se dovedește algoritmul de împărțirea a polinoamelor, algoritmii utilizat pentru operațiile de adunare, scădere, înmulțire, derivare și integrare având la bază doar operații aritmetice simple asupra monoamelor ce compun operanzii.

Algoritmul de împărțire este compus din următorii pași:

- Se ordonează monoamele celor două polinoame în ordinea descrescătoare a gradelor;
- Se împarte primul monom al polinomului deîmpărțit la primul monom al împărțitorului;
- Se înmulțește câtul cu împărțitorul, iar rezultatul înmulțirii se scade din deîmpărțit, obținându-se astfel restul împărțirii;
- Se repetă pașii anteriori, considerând restul ca noul deîmpărțit, până când gradul restului devine mai mare decât gradul împărțitorului.

4. Implementare

4.1. Clase și metode definite

Denumirile claselor, ale variabilelor instanță și ale metodelor ce urmează a fi sintetizate și prezentate în cele ce urmează sunt sugestive, nefiind necesară explicitarea semnificației acestora.

a. Pachetul model

- **Clasa Monomial** modelează conceptul matematic de monom prin variabilele instanță private **coefficient**, de tip double întrucât la operațiile de împărțire și integrare se pot obține și coeficienți reali, nu doar întregi, și **degree**, metodele cele mai semnificative ale aceste clase fiind:

Antetul metodei	Utilizarea
`String toString()`	- afișarea rezultatelor în interfața grafică sub formă de string-uri
`Monomial multiplyMonomial(Monomial m)`	- înmulțirea a două monoame
`Monomial derivative()`	- derivarea unui monom
`Monomial integrateMonomial()`	- integrarea unui monom
`Monomial divideMonomial(Monomial m)`	- împărțirea a două monoame
`static Monomial parseMonomial(String mon)`	- parsarea unui monom sub forma unui string pentru a obține coeficientul și gradul monomului (utilizată la transformarea string-urilor introduse de utilizator în interfața grafică în obiecte de tip Monomial)

- **Clasa Polynomial** modelează conceptul de polinom, sub forma listei sale de monoame, **monomials**, definind astfel tipul obiect asupra căruia calculatorul va efectua operațiile necesare satisfacerii cerințelor funcționale. Metodele semnificative ale clasei Polynomial sunt:

Antetul metodei	Utilizarea
`int getPolynomialDegree()`	- determinarea gradului polinomului, ca fiind egal cu maximul dintre gradele monoamelor din lista de monoame
`Polynomial reduceTermsOfSameDegree()`	- aducerea polinomului la forma simplificată, efectuând calculele posibile între monoamele de același grad
`int getValue(int x)`	- determină valoarea polinomului în punctul transmis ca parametru
`boolean isNull()`	- verifică dacă un polinom este polinomul nul (dacă lista sa de monoame este vidă sau dacă gradul acestuia este 0 și coeficientul rezultat în urma reducerilor este și el 0)
`String toString()`	- afișarea rezultatelor în interfața grafică sub formă de string-uri
`void sortDescendingDegree()`	- sortarea monoamelor din componența polinomului în ordinea descrescătoare a gradelor

- Clasa Operations modelează cele șase operații definite în cadrul cerințelor funcționale ca metode statice:

Antetul metodei	Utilizarea
`static Polynomial add(Polynomial p, Polynomial q)`	- adunarea celor două polinoame transmise ca parametru; rezultatul returnat este tot un polinom
`static Polynomial sub(Polynomial p, Polynomial q)`	- scăderea celor două polinoame transmise ca parametru; rezultatul returnat este tot un polinom

`static Polynomial multiply(Polynomial p, Polynomial q)`	- înmulțirea celor două polinoame transmise ca parametru; rezultatul returnat este tot un polinom
`static ArrayList<Polynomial> divide(Polynomial p1, Polynomial p2)`	- împărțirea celor două polinoame transmise ca parametru; rezultatul returnat este o listă de două polinoame, primul reprezentând câtul împărțirii, iar cel de-al doilea reprezentând restul
`static Polynomial derivative(Polynomial p)`	- derivata polinomului transmis ca parametru; rezultatul este tot un polinom
`static Polynomial integrate(Polynomial p)`	- integrala polinomului transmis ca parametru; rezultatul este tot un polinom

- Clasa **DegreeComparator** modelează un comparator pentru sortarea monoamelor în ordinea descrescătoare a gradului, prin implementarea interfeței **Comparator** și suprascrierea metodei ***`int compare(Monomial o1, Monomial o2)`***.

- Clasa **CalcModel** implementează modelul aplicației, prin variabilele instanță **Polynomial p**, **Polynomial q** și **ArrayList<Polynomial> res**. Rezultatul este o listă de polinoame pentru a putea acoperi cazul în care se efectuează o operație de împărțire, când primul polinom din lista res va reprezenta câtul împărțirii, fiind returnat sub formă de string cu ajutorul metodei ***`String getResult()`*** iar cel de-al doilea polinom restul împărțirii, care va fi accesibil prin apelul metodei ***`String getRemainder()`***. Metodele ***`void addPolynomials()`***, ***`void subPolynomials()`***, ***`void mulPolynomials()`***, ***`void divPolynomials()`***, ***`void difPolynomial()`***, ***`void intPolynomial()`*** efectuează cele șase operații asupra variabilelor instanță p și q ale clasei, prin apelul metodelor statice din clasa **Operations**, cu mențiunea că pentru operațiile cu un singur operand, acesta va fi considerat ca fiind variabila instanță p. Alte metode specifice clasei **CalcModel** sunt ***`void reset()`***, folosită la resetarea modelului în vederea efectuării de noi operații pe alte valori ale variabilelor instanță de intrare, și metoda ***`void resetResult()`*** folosită pentru reinițializarea variabilei instanță res, în vederea efectuării unei alte operații pe aceleași valori ale variabilelor p și q.

b. Pachetul view

- Clasa **CalcView** este responsabilă de realizarea interfeței grafice cu utilizatorul, având drept variabile instanță diferitele butoane, etichete și câmpuri de text ce pot fi vizualizate de către utilizator la folosirea aplicației. Metoda ***`void reset()`*** aduce interfața grafică în starea inițială, caracterizată prin faptul că atât câmpurile de text corespunzătoare operanzilor, cât și cel corespunzător rezultatului să nu conțină text, iar metoda ***`void showError(String errMessage)`*** permite apariția unei ferestre ce afișează un mesaj de eroare în cazul apariției unei excepții de tipul **InvalidPolynomialException** sau **DivisionByZeroException**. De asemenea, clasa **CalcView** conține numeroase metode de setare și accesare a textului din cele trei câmpuri de text, dar și metodele necesare adăugării ascultătorilor definiți în cadrul claselor interne din clasa **CalcController** celor șase butoane din interfața grafică a aplicației.

c. Pachetul controller

- Subpachetul **controller.exceptions** conține clasele excepție **InvalidPolynomialException** care semnalează faptul că unul dintre polinoamele introduse de către utilizator nu este valid (nu respectă forma generală $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$, unde a_i, $i = 1, n$ sunt numere

întregi) și **DivisionByZeroException** care semnalează faptul că se dorește efectuarea unei operații de împărțire, iar cel de-al doilea polinom introdus, care reprezintă în acest caz împărțitorul, este polinomul nul.

- Clasa **Validity** a fost implementată în scopul verificării validității datelor introduse de către utilizator, prin definirea a trei pattern-uri care să acopere cele trei cazuri care pot apărea la introducerea unui monom sub formă de string: monom în cazul general (cu coeficient și exponent indicat explicit), monom de gradul 1 și monom de gradul 0. Pentru definirea pattern-urilor și verificarea validității expresiilor s-au folosit expresii regulate, prin importarea pachetului java.util.regex.*. Metodele semnificative pentru această clasă:

Antetul metodei	Utilizarea
`static boolean isValidMonomial(String mon)`	- verifică dacă string-ul trimis ca parametru reprezintă un monom valid
`static ArrayList<String> splitToMonomials(String s) throws InvalidPolynomialException`	- verifică dacă string-ul trimis ca parametru reprezintă un polinom valid, caz în care returnează o listă de string-uri reprezentând monoamele din care este alcătuit polinomul introdus de utilizator sau, în cazul introducerii unui polinom invalid, aruncă o excepție de tipul **InvalidPolynomialException**; în cadrul acestei metode au fost folosite, conform specificației problemei, expresii regulate pentru extragerea monoamelor din polinomul introdus ca șir de caractere

- Clasa **CalcController** reprezintă controllerul aplicației, fiind definit cu ajutorul variabilelor instanță **CalcModel model** și **CalcView view**. Această clasă este responsabilă de preluarea informațiilor introduse de către utilizator în interfața grafică a aplicației și translatarea lor în modelul obiectual asociat modelului matematic al problemei, în vederea efectuării operațiilor cerute. Clasele interne **AddListener, SubListener, MulListener, DivListener, DifListener. IntListener** și **ClearListener** implementează interfața **ActionListener** pentru a permite definirea de ascultători corespunzători butoanelor asociate celor șase operații și butonului de resetare din interfața grafică, prin suprascrierea metodei `void actionPerformed(ActionEvent e)`. Metodele `void parseInputP() throws InvalidPolynomialException` și `void parseInputQ() throws InvalidPolynomialException` realizează parsarea celor două string-uri introduse de utilizator și transpunerea acestora în polinoamele ce caracterizează modelul, cu eventualitatea aruncării unei excepții de tipul **InvalidPolynomialException** în cazul introducerii unui string care să nu reprezinte un polinom valid.

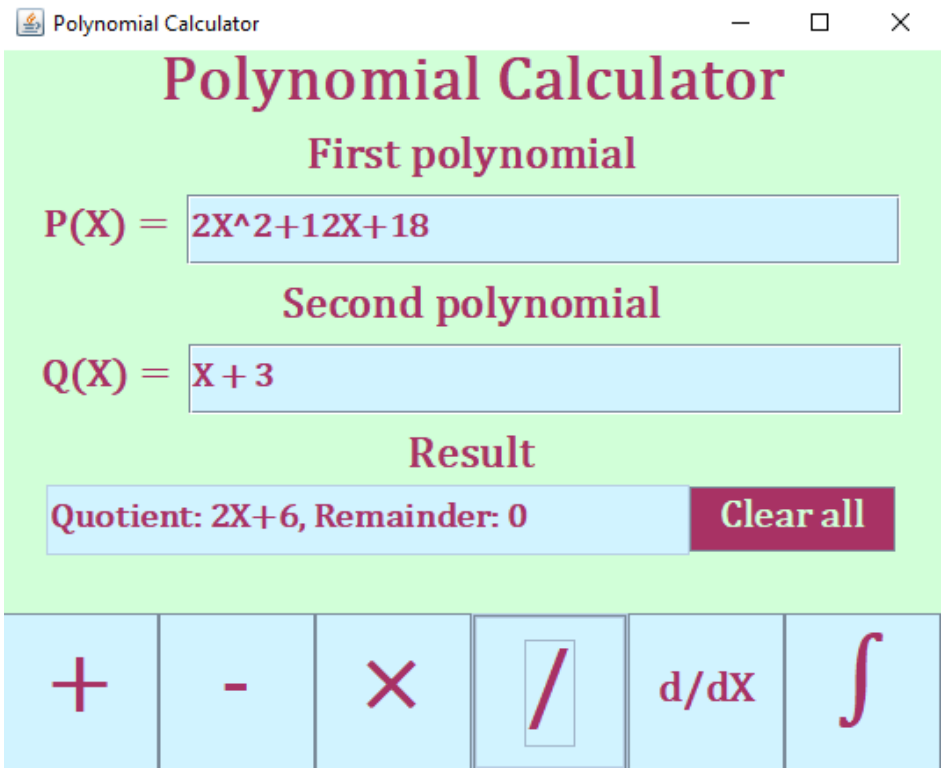
d. Pachetul calculator

- Clasa **CalcMVC** instanțiază într-o metodă `main` cele trei componente ale unui calculator de polinoame realizat în arhitectura MVC (modelul, vizualizarea și controllerul) pentru a permite rularea aplicației dezvoltate.

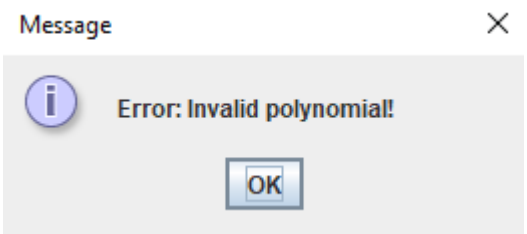
4.2. Interfața grafică cu utilizatorul (GUI)

Aplicația dispune de o interfață grafică intuitivă, dezvoltată folosind Swing și fiind construită pe baza unei JFrame. Aceasta are un panou principal în care au fost adăugate diferite subpanouri, implementate folosind JPanel, conținând componentele necesare unei aplicații de tip calculator.

Astfel, interfața grafică are în componența sa două câmpuri de text editabile pentru introducerea polinoamelor de către utilizator, cel de-al treilea, needitabil, fiind rezervat afișării rezultatului operației efectuate. Etichetele sunt realizate folosind JLabel, în timp ce câmpurile de text au la bază componente de tip JTextField. De asemenea, în interfața grafică a aplicației se pot regăsi șase butoane, câte unul rezervat fiecărei operații de efectuat, și butonul suplimentar de resetare, „Clear all". Interfața grafică facilitează folosirea calculatorului de polinoame de către orice tip de utilizator și oferă o interacțiune mai bună a utilizatorului modelul obiectual al soluției, putând fi vizualizată în figura de mai jos. De menționat este faptul că în cazul operației de împărțire, atât câtul, cât și restul împărțirii va fi afișat în același câmp de text, specificându-se care este câtul și care este restul.



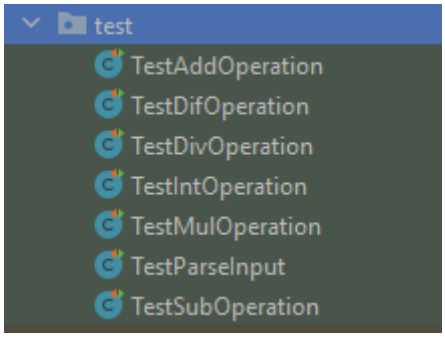
Aplicația semnalizează o eroare prin apariția unei ferestre cu un mesaj specific în cazul în care utilizatorul a introdus un polinom invalid sau se încearcă împărțirea la 0.



5. Rezultate

Testarea modelului, și anume, a operațiilor asupra polinoamelor a fost efectuată folosind **JUnit**, fiind scrise câte cinci teste pentru fiecare operație necesară satisfacerii cerințelor funcționale ale problemei. Teste scrise încearcă să surprindă toate cazurile posibile în care modelul aplicației s-ar putea găsi la efectuarea unei operații din setul de care acesta dispune. S-a verificat stocarea și afișarea corectă a coeficienților în cazurile speciale în care aceștia sunt -1 și +1, dar și corectitudinea cazurilor în care exponentul este 1 sau 0. De asemenea, a fost testată parsarea polinoamelor în vederea asigurării

corectitudinii transformării șirului de caractere introdus de utilizator în interfața grafică a aplicației în modelul obiectual ce definește polinoamele asupra căruia se efectuează operațiile definite. Și în ceea ce privește parsarea au fost introduse cazurile speciale de test, expuse mai sus, rezultatul testelor fiind unul pozitiv. Toate testele efectuate au fost finalizate cu succes, atât pentru scenariile obișnuite, cât și pentru cele speciale, menționate anterior, clasele de test fiind vizibile în următoarea captură de ecran:



6. Concluzii

Concluzionând, se poate afirma că această temă vine să soluționeze problema efectuării manuale a operațiilor pe polinoame, fiind implementată conform paradigmei programării orientate pe obiect și proiectându-se ca o mini-aplicație ce poate fi folosită de orice tip de utilizator, dispunând de un pattern arhitectural care permite separarea modelului matematic, obiectual, de interfața grafică cu care utilizatorul interacționează. Pentru dezvoltatorul proiectului, implementarea acestuia s-a dovedit utilă, având drept rezultat o mai bună însușire a manipulării și dezvoltării aplicațiilor Object Oriented, prin exersarea implementării practice a conceptelor specifice aceste paradigme de programare, dar și acumularea de noi cunoștințe despre testarea unitară a unui program și despre folosirea expresiilor regulate în determinarea tiparului pe care un text sau șir de caractere îl urmărește.

Printre posibilele dezvoltări ulterioare ale aplicației se numără introducerea unei noi file în care să se rețină istoricul operațiile și calculelor efectuate și setarea câmpurilor de text astfel încât, în timpul introducerii polinoamelor asupra cărora se urmărește efectuarea de operații, dacă se sesizează că datele introduse nu respectă pattern-ul stabilit, fundalul acestora să devină roșu pentru a atenționa utilizatorul cu privire la faptul că introduce date invalide

7. Bibliografie

- ASSIGNMENT_1_SUPPORT_PRESENTATION (furnizat pe Microsoft Teams)
- Cursuri și lucrări de laborator de la disciplina Programare Orientată pe Obiecte, studiată în semestrul I al anului universitar 2020-2021
- https://docs.oracle.com/javase/8/docs/api/index.html
- https://en.wikipedia.org/wiki/Polynomial_long_division
- https://docs.oracle.com/javase/tutorial/uiswing/
- https://www.vogella.com/tutorials/JUnit/article.html
- https://docs.oracle.com/javase/tutorial/essential/regex/