**Documentație Tema 4 Baciulescu Raul, gr. 236**

**Enuntul problemei:**

Se considera **n** polinoame reprezentate prin lista de monoame.

Se cere adunarea polinoamelor folosind o implementare multithreading (p threaduri).

Consideratii generale:

- reprezentarea unui polinom in memorie: lista inlantuita (1 nod=1monom) ordonata dupa exponentii monoamelor cu urmatorul INVARIANT (predicat adevarat la orice moment al executiei) de reprezentare:

-monoamele sunt ordonate dupa exponenti

-nu se pasteaza in lista monoame cu coeficient 0;

- nu exista doua noduri (monoame) cu acelasi exponent

- polinoamele se citesc din fisiere – cate un fisier pentru fiecare polinom;

- un fisier contine informatii de tip (coeficient, exponent) pentru fiecare monom al

unui polinom,

- fisierele input se creeaza prin generare de numere aleatoare.

(Conditie: fisierele nu contin monoame cu coeficient egal cu 0 dar nu sunt ordonate dupa exponent!)

Rezolvare:

Se porneste prin crearea unei liste inlantuita - L corespunzatoare unui polinom nul.   
In final aceasta lista va continue polinomul rezultat.

Metoda A) Implementare secventiala

* Se citeste pe rand din fiecare fisier cate un monom si se adauga in lista rezultat -L (atentie – invariantul trebuie sa ramana adevarat dupa fiecare adaugare de monom).

Metoda B) Implementare paralela – p threaduri

1. Primul thread citeste cate un monom si il adauga intr-o structura de date de tip coada.

(conditie – pentru structura de tip coada NU se admite folosirea unei structuri de date pentru care partea de sincronizare este deja implementata!!!)

1. Celelalte threaduri preiau cate un monom din coada si il aduna la polinomul reprezentat in lista L.
   * Se continua operatiile 1., 2. pana cand toate monoamele, din toate fisierele, sunt adunate la lista L.
2. Primul thread scrie rezultatul obtinut in lista L intr-un fisier rezultat

(conditie: fisierul nu contine monoame cu coefficient egal cu 0)

**Detalii de implementare:**

Graphical user interface

Description automatically generated

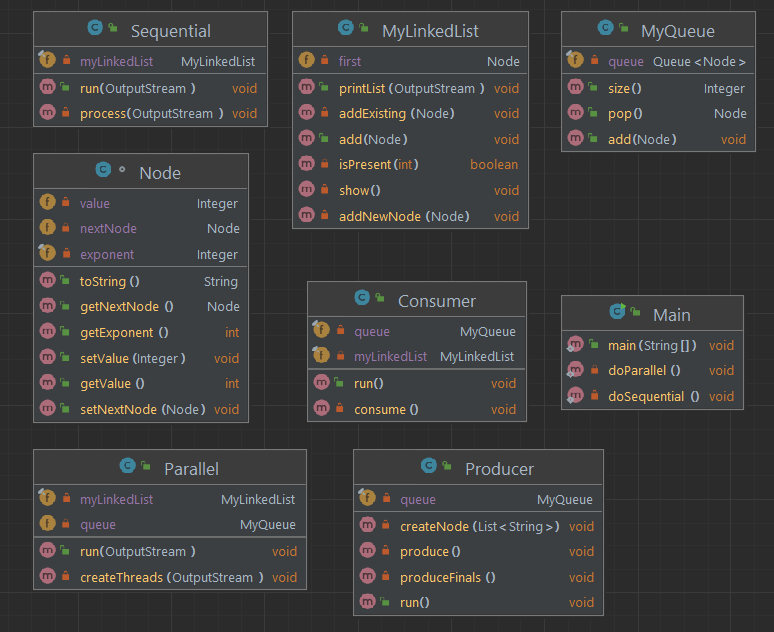
Am folosit clasele utilitare CreateFile, care creaza un fisier de output, daca acesta nu exista, PolynomialGenerator care genereaza un polinom in functie de mai multi parametri si clasa Constants care contine anumite variabile cheie programului. PolynomialGenerator genereaza polinoame random in functie de variabilele maxMono, numberOfPolynomials si maxDegree si le scrie in fisiere separate. Un fisier de polinom are urmatorul format

value1 exponent1

value2 exponent2

…

valuen exponentn



Pentru implementarea secventiala, am folosit clasa Sequential, care prin functia process citeste pe rand din fiecare fisier de polinoame, si adauga fiecare monom la lista MyLinkedList.

Clasa MyLinkedList contine implementarea unei liste inlantuite care incepe de la nodul first.

La fiecare adaugare a unui nod verific daca exponentul acestuia exista in lista. Daca exista ii caut exponentul corespunzator si cresc valoarea, daca nu creez un nod nou, pastrand lista ordonata descrescator.

Pentru implementarea paralela am folosit sablonul Producer Consumer. Clasa Parallel este responsabila de crearea thread-urilor, listei inlantuite cu rezultatul si a coadei. Coada MyQueue este o coada sincronizata, folosita de Producer. Producer citeste fiecare fisier de polinoame si adauga monoamele sincronizat. Sincronizarea este facuta adaugand cuvantul synchronized la antetul functiilor din MyQueue si MyLinkedList. Functia pop() din MyQueue contine si un apel la wait(), atunci cand coada este goala. Pop() isi continua executia atunci cand este adaugat ceva in coada si se apeleaza functia notifyAll(). Ca sa marches ca nu mai exista nimic de executat de consumer, adaug null-uri in coada, atatea null-uri cate consumer am. Cand un consumer citeste un null, inseamna ca acel consumer trebuie sa se opreasca pentru ca nu mai exista monoame de procesat. Clasa Parallel creaza thread-urile, un Producer si mai multe Consumere.

**Teste:**

1. 10 polinoame fiecare cu gradul maxim 1000 si cu maxim 50 monoame
   1. p = 4: 53.7
   2. p = 6: 53
   3. p = 8: 51.5
   4. secvential: 53.8
2. 5 polinoame fiecare cu gradul maxim 10000 si cu maxim 100 monoame
   1. p = 4: 53.8
   2. p = 6: 53.5
   3. p = 8: 53.5
   4. secvential: 52

* se observa la primul exemplu ca timpul paralel cu 8 thread-uri este mai rapid decat restul
* alte diferente nu se observa.