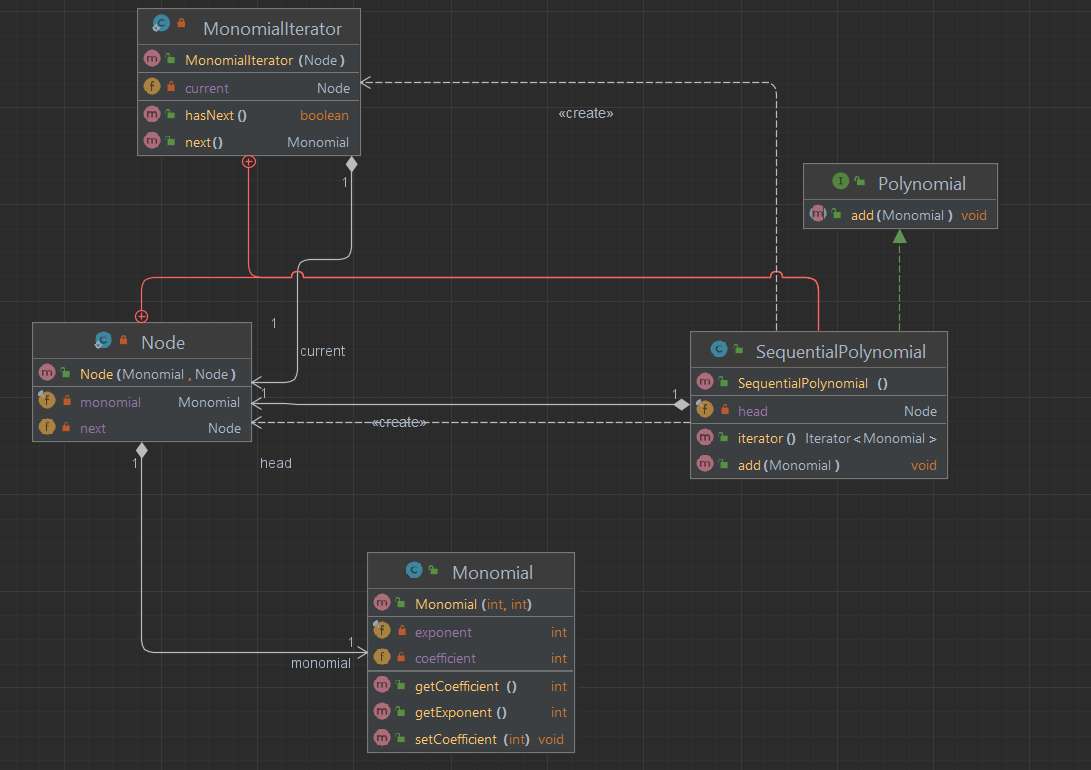
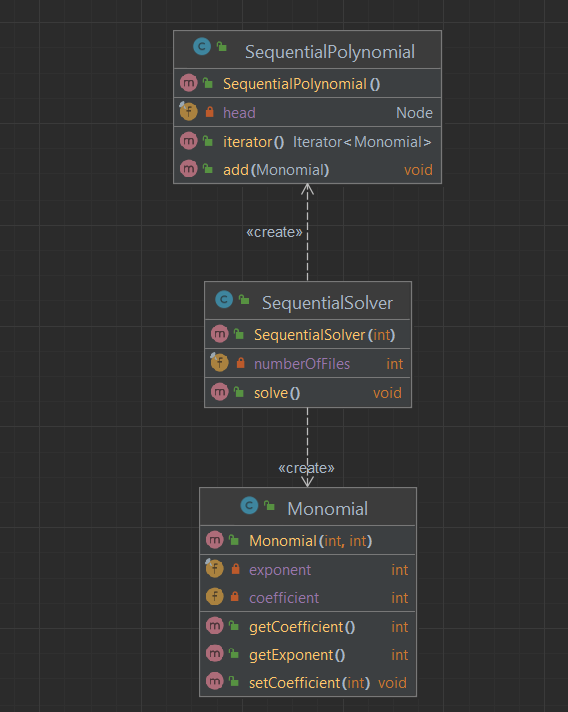
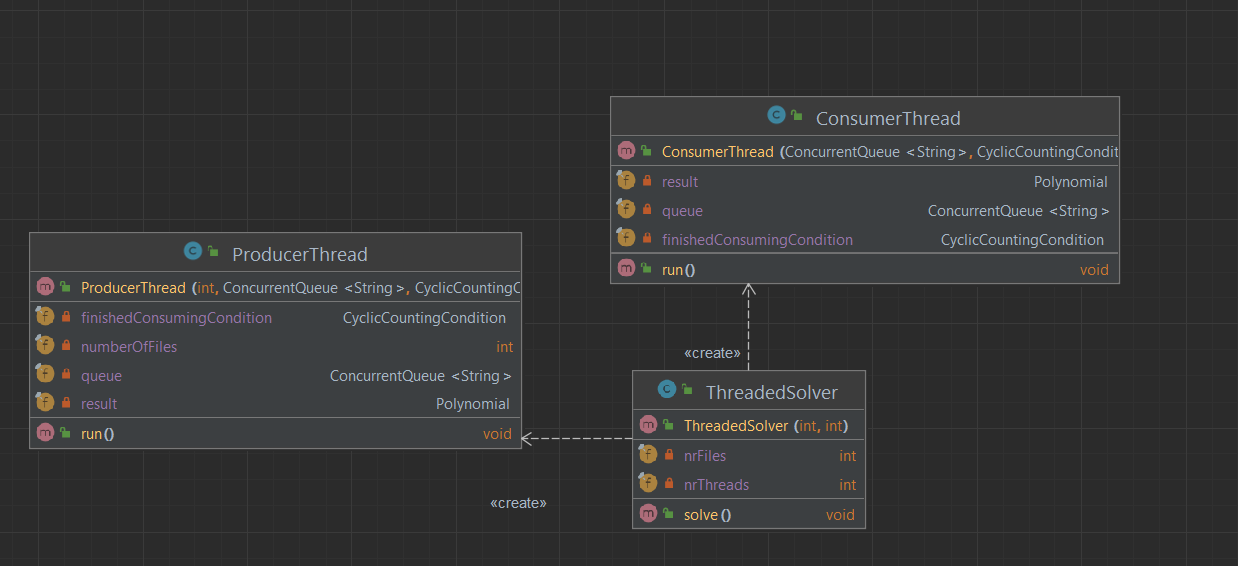
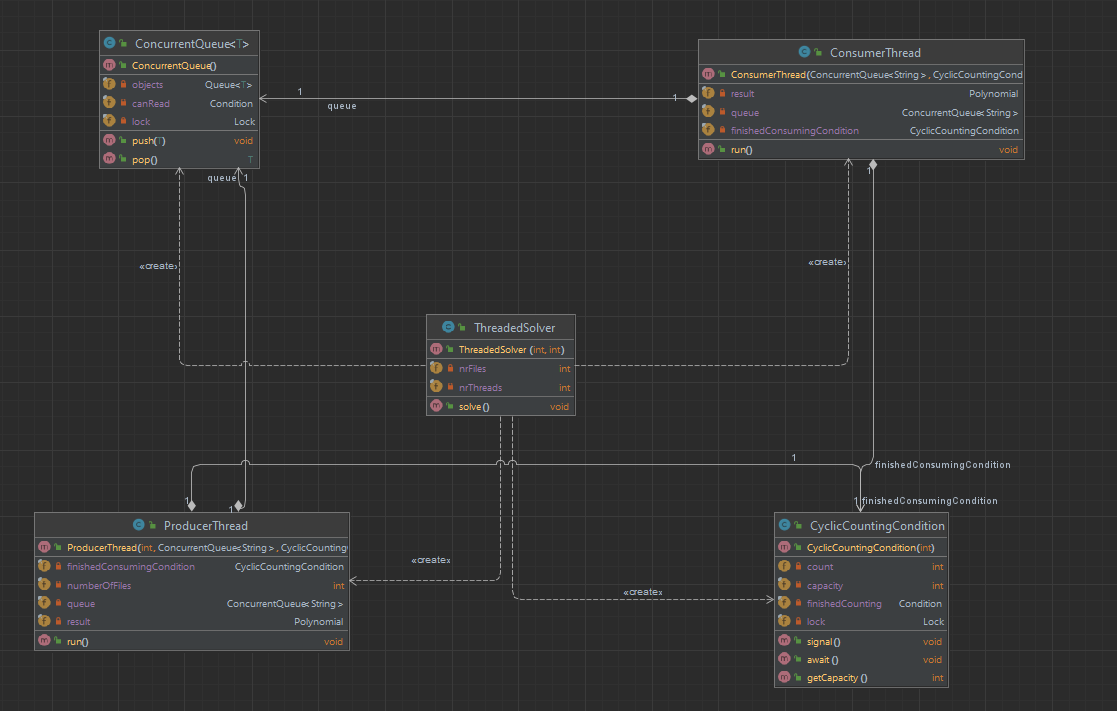
Documentatie PPD Lab 4

* Info:
  + OS: Windows 10
  + Processor: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz
  + Cores: 8
  + Logical Processors: 16
  + RAM: 16.0 GB (15.7 GB usable)
* Idee:
  + Fisierele de intrare au pe fiecare linie un string ce reprezinta un monom – un monom este de forma “coeficient exponent”
  + Producer thread citeste din fisiere si scrie monoamele intr-o coada de asteptare sincronizata
  + Atunci cand producer-ul termina de citit, pushuieste in coada atatea valori de null cati consumers sunt
  + Consumer threads citesc din coada monoamele
  + Daca se citeste null, inseamna ca nu mai sunt elemente de citit
  + Dupa ce un monom valid este citit, acesta este adaugat in obiectul rezultat de tip Polinom
  + Odata ce Consumers termina de procesat coada, acestia semnaleaza asta
  + Dupa ce toti consumers semnaleaza ca au terminat de citit, producer scrie in fisier rezultatul
* Specificatii:
  + Specifcatiile sunt puse pe codul sursa, la fiecare metoda, impreuna cu detaliile extra de implementare
* Java:
  + Diagrame:

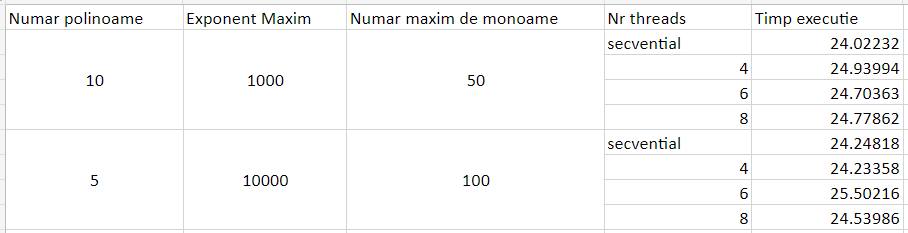








* + Rezultate:



* + Pentru toate cazurile:
    - Varianta secventiala este relativ cea mai eficienta
    - Cu cresterea numarului de thread-uri avem o distributie relativ normala – pentru cazul cu 10 polinoame, 6 thread-uri este un punct de minim,, iar pentru 5 este de maxim
    - Ca si explicatie pentru performanta, pentru varianta threaded, polinomul nu este sincronizat la maxim – sincronizarea s-ar putea face la nivel de nod, nu de lista
* Overall Conclusions:
  + Varianta secventiala castiga la aceasta implementare, probabil deoarece sincronizarea polinomului se face la nivel de obiect, nu de nod