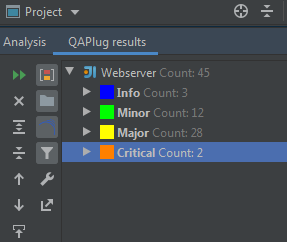
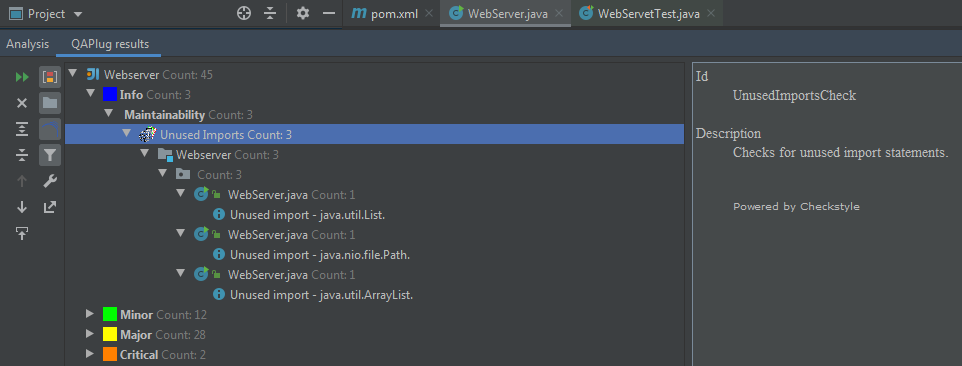
Analiza statica si dinamica

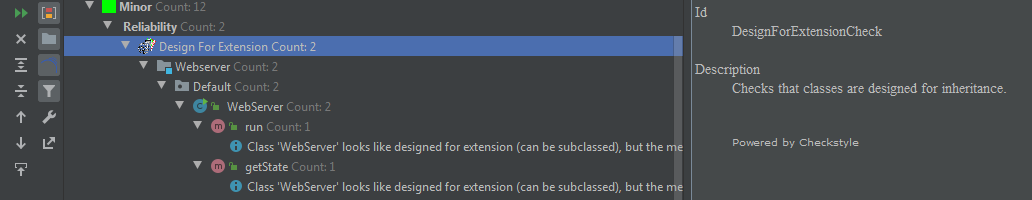
1. Analiza statica

Am folosit plugin-ul QAPlug in IntelliJ care contine 3 tool-uri pentru analiza statica: FindBugs, Checkstyle si PMD. Daca foloseam doar FindBugs aveam maxim 2 bug-uri, asa imi gaseste 45.

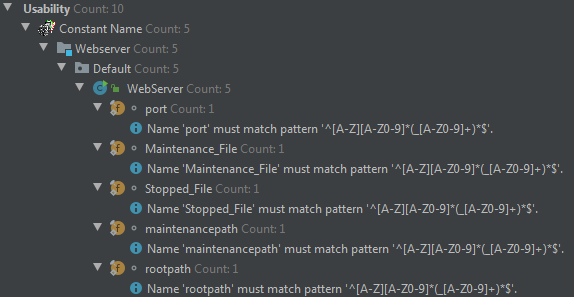




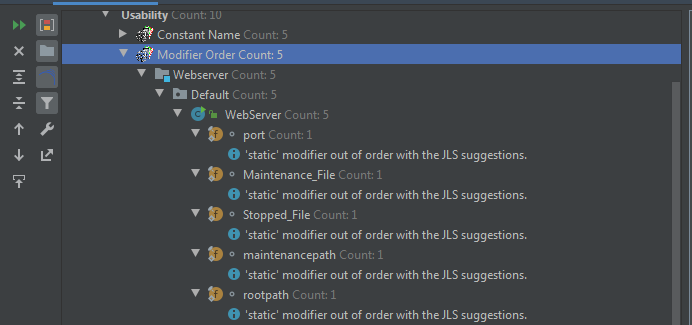
Primele 3 probleme sunt semnalizate de Checkstyle si se refera la maintainability. Sunt 3 importuri pe care nu le folosesc niciunde in aplicatia mea (java.util.List, java.util.ArrayList, java.nio.file.Path). Aceste probleme apartin de categoria Info, deci sunt mai mult informative pentru a scrie un corect cat mai curat.

Urmatoarele 12 probleme apartin de categoria Minor, detectate tot de Checkstyle. 

In poza de mai sus sunt 2 probleme care tin de Reliability. Deoarece clasa Webserver nu contine cuvantul cheie final, Checkstyle considera ca se vrea mostenirea ei. Metodele run(), respectiv getState() au deja o implementare si atunci nu se va putea face mostenire. Solutia ar fi sau sa fac clasa Webserver final sau sa pun cuvantul cheie abstract/static/final celor 2 metode.

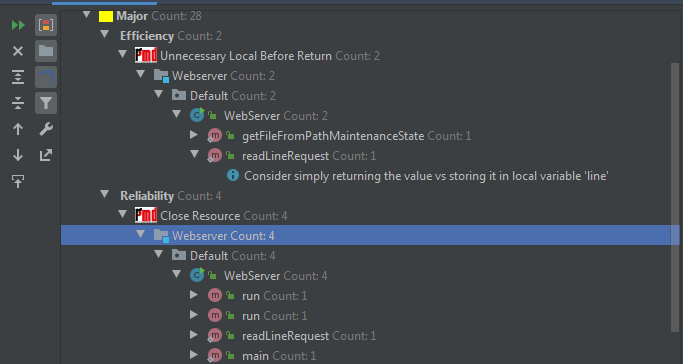


Cele 5 probleme de mai sus se refera la faptul ca numele constantelor ar trebui sa match-uie expresia regulata: '^[A-Z][A-Z0-9]\*(\_[A-Z0-9]+)\*$' .



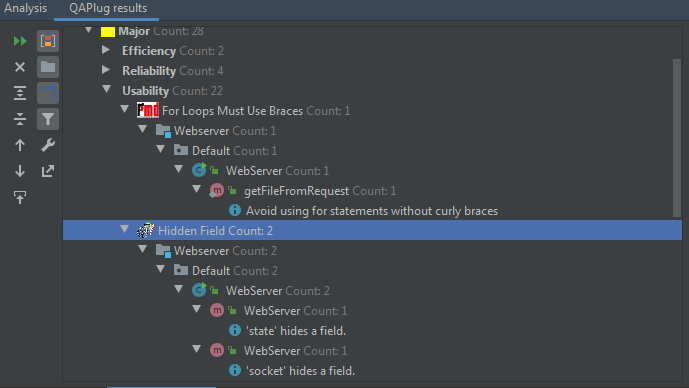
Mai sus sunt 5 probleme care tin de ordinea modificatorilor, care conform Java Language specification ar trebui sa fie urmatoarea: public, protected, private, abstract, static, final, transient, volatile, synchronized, native, strictfp. La mine constantele au ordinea final static in loc de static final, de aceea apar cele 5 bug-uri.

Urmeaza 28 de bug-uri din categoria Major.

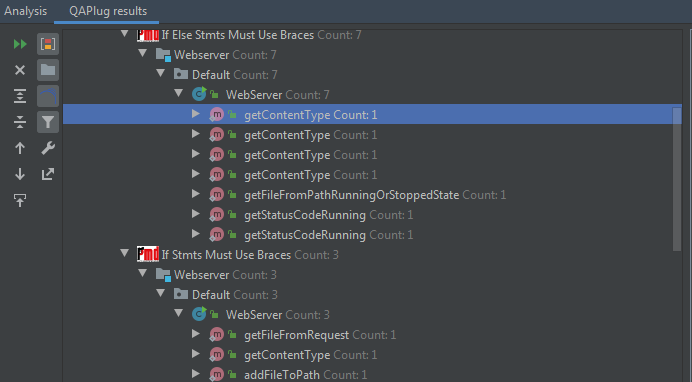


Primele 2 tin de eficienta si se refera la faptul ca in functiile readLineRequest, respectiv getFileFromPathMaintenanceState puteam sa returnez direct, nu sa mai folosesc o variabila locala in care sa retin ceea ce trebuie returnat pe care mai apoi sa o returnez.

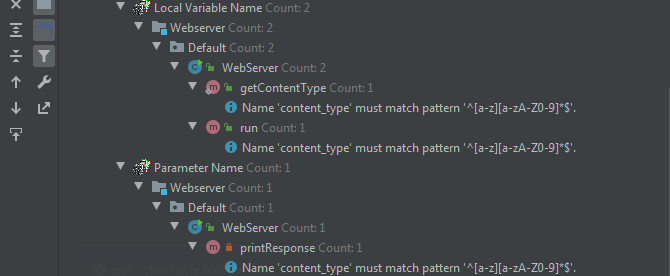
Urmatoarele 4 tin de reliability si se refera la inchiderea resurselor folosite, de exemplu ServerSocket, BuferredReader, PrintWriter. Aceste 6 bug-uri sunt semnalate de PMD.



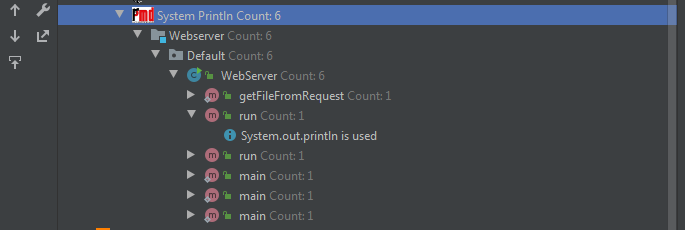
Urmeaza 22 de bug-uri care tin de usability. La primul mi se semnaleaza ca folosesc un for fara acolade. Deci chiar daca am o singura instructiune pentru for va trebui sa folosesc acolade. Urmatoarele 2 bug-uri imi semnaleaza ca variabilele locale ascund campuri ale clasei, deoarece au acelasi nume cu campurile. Acest lucru se intampla chiar in constructor la instructiunile this.socket = socket; respectiv this.state = state .



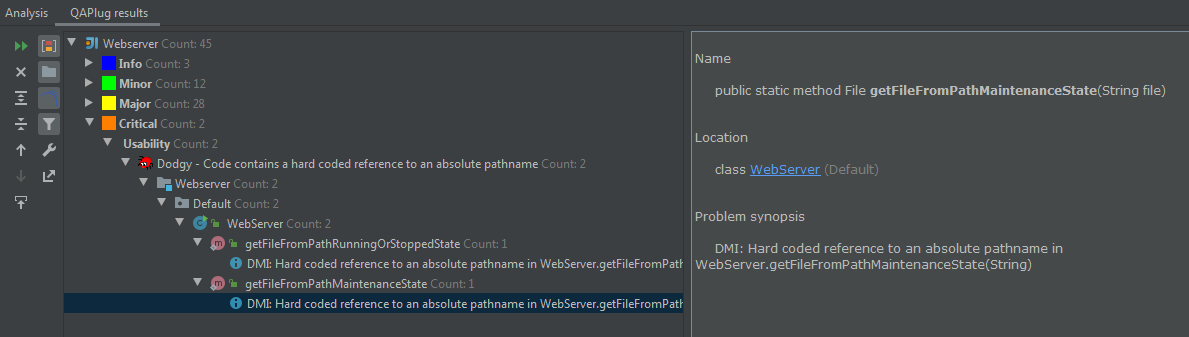
Mai sus am 10 bug-uri deoarece din nou nu am folosit acolade, insa de data aceasta la if-uri .



In imaginea de sus din nou 3 probleme legate de data aceasta de numele parametrilor si al variabilelor locale care trebuie sa match-uie expresia regulata: '^[a-z][a-zA-Z0-9]\*$'.



Mai sus sunt 6 bug-uri care se leaga de faptul ca folosesc "System.(out|err).print" pentru a afisa informatie care sa ma ajute la debugging, in loc sa folosesc un Logger pentru acest lucru.



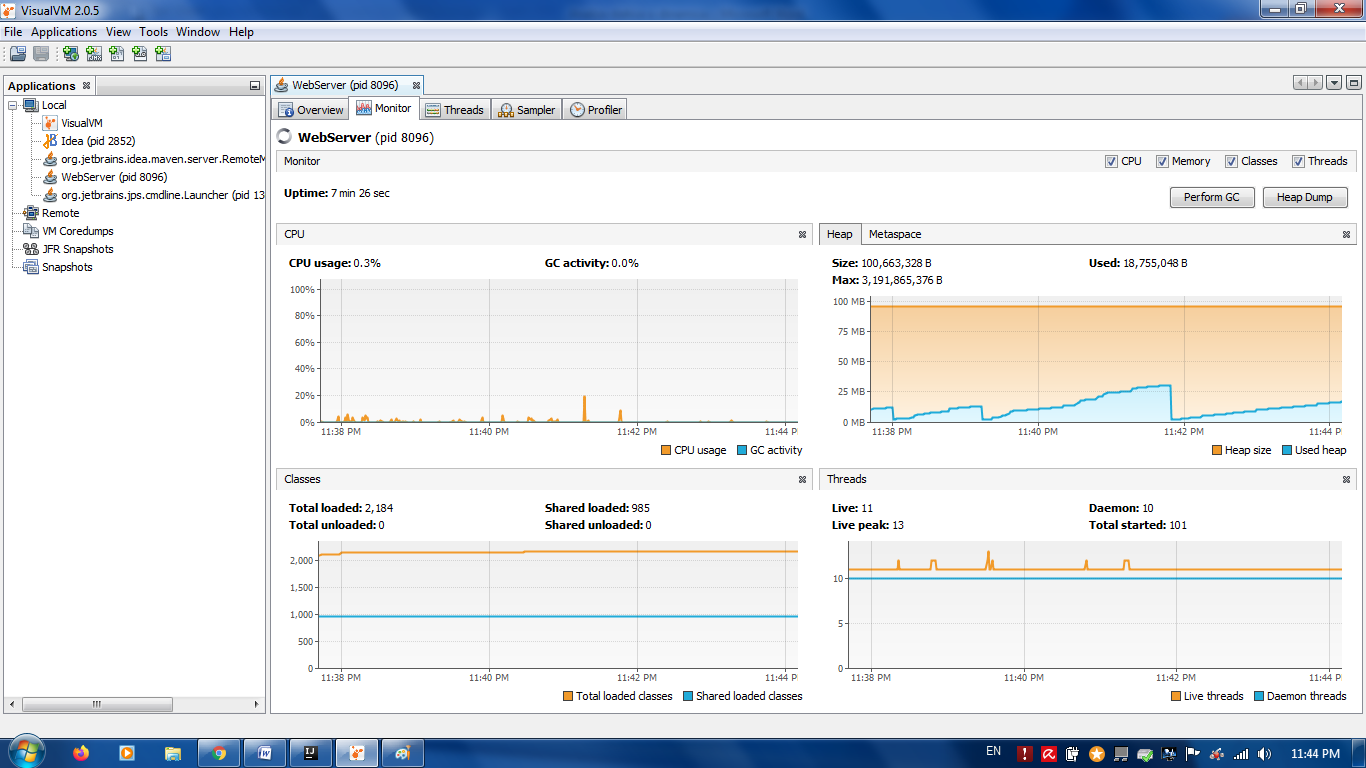
Ultimele 2 bug-uri sunt semnalate de FindBugs si fac parte din categoria Critical. Acestea se refera la faptul ca in cele 2 functii din imaginea de mai sus folosesc referinte hard codate la un path absolut

( *maintenancepath*="C:\\Users\\Daniel\\Desktop\\Huple\_Razvan\\Week\_01\\maintenance\\",

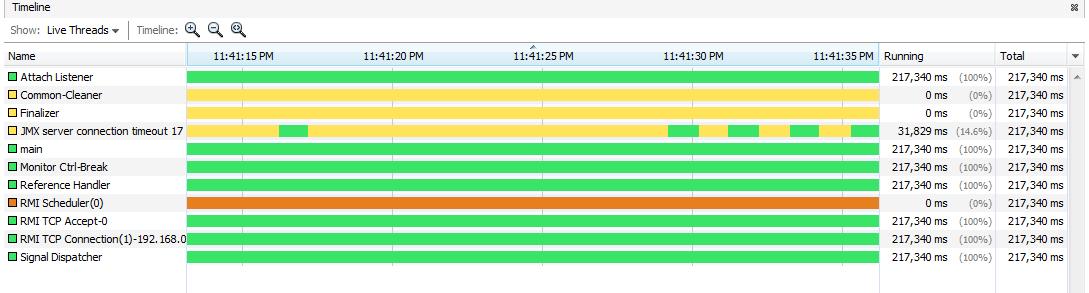
*rootpath*="C:\\Users\\Daniel\\Desktop\\Huple\_Razvan\\Week\_01\\client\\" ). Solutia ar fi sa mut aceste fisiere undeva in interiorul proiectului, si atunci voi putea folosi un path relativ. In acest fel aplicatia ar putea fi utilizata si de alti utilizatori. Folosind path-ul absolut, ceilalti useri care ar utiliza aplicatia ar fi nevoiti sa-si creeze efectiv acel path ca aplicatia sa mearga, ceea ce e destul de absurd.

1. Analiza dinamica

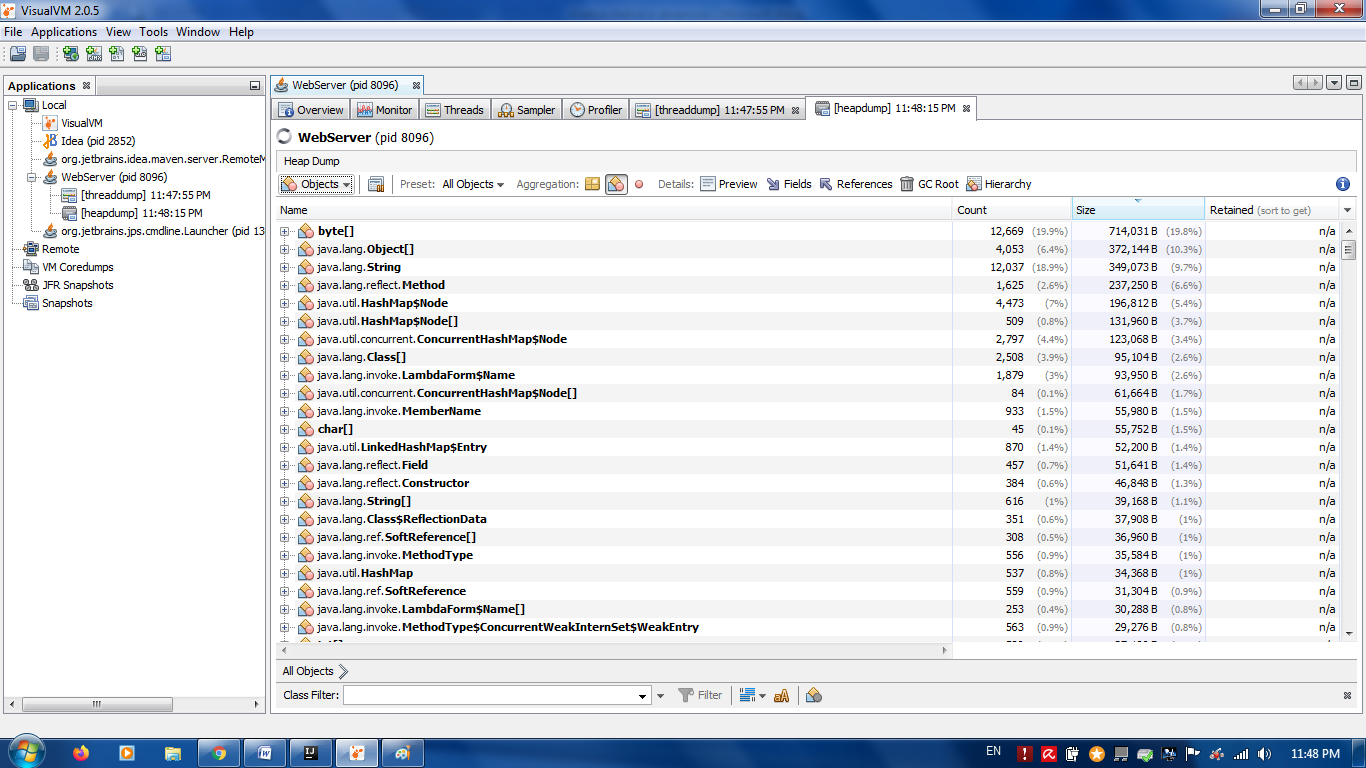
Pentru analiza dinamica am folosit tool-ul VisualVm care ofera diferite informatii si grafice despre aplicatia java la runtime.

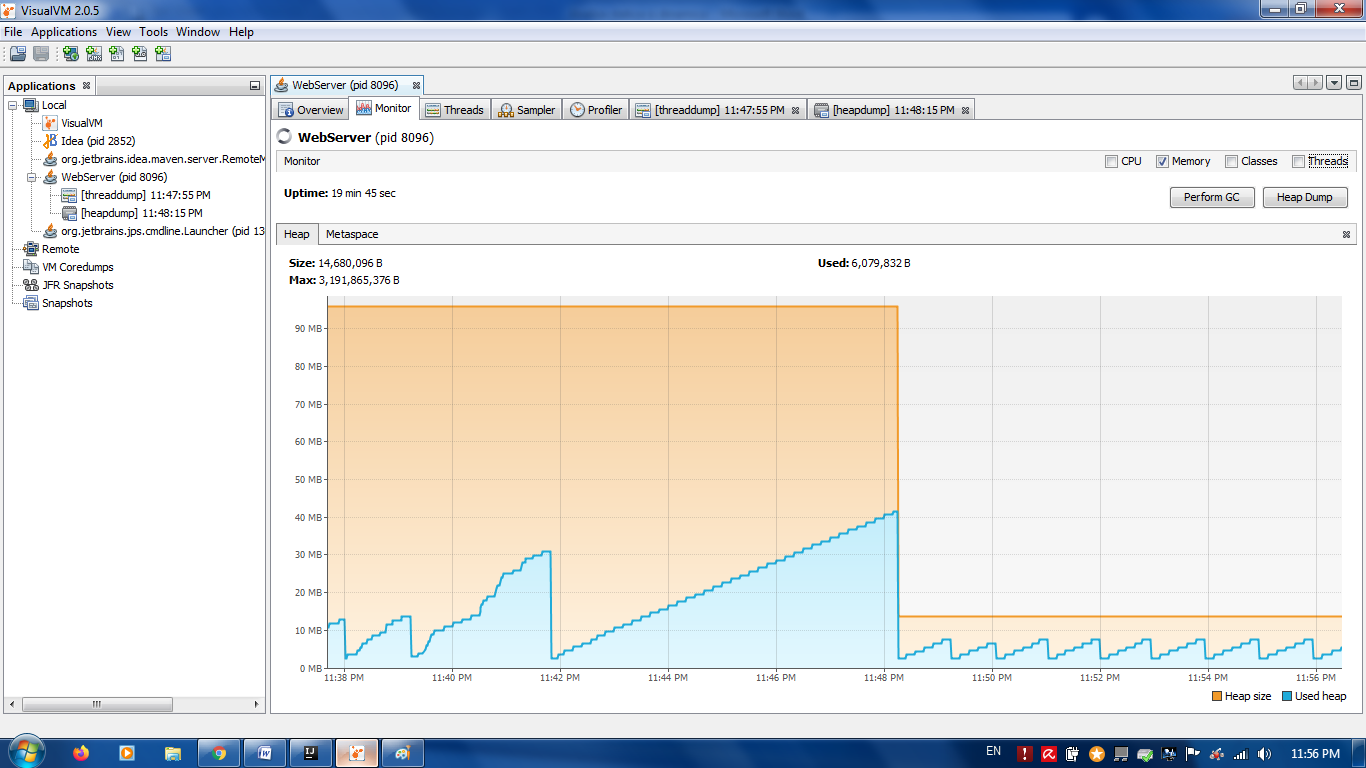


Am un numar de 2184 clase, din care 985 sunt share loaded. In graficul cu threads, apar acele "varfuri", care indica practic ca utilizatorii navigheaza pe site.



Singurul user thread este main, restul fiind daemon threads. Main-ul are starea running.

Numarul de obiecte byte este cel mai mare si ocupa cea mai multa memorie. Cred ca se intampla din cauza ca stringuri-le sunt luate ca bytes. Iar la citirea cu buffered reader respectiv printarea cu printwriter se proceseaza multe stringuri din fisiere.



Asa arata heap-ul. Descresterea de dupa 11:48 se datoreaza heapdump-ului care face GC.