Analiza statica

In urma executarii analizei statice, analizoarele utilizate au produs urmatoarele rezultate:

1. FindBugs: 0
2. Checkstyle: 229

Findbugs:

Detecteaza posibilele buguri in programe Java. Potentialele erori sunt clasificate in patru categorii:

1. Scariest
2. Scary
3. Troubling
4. Of concern.

In cazul de fata, Findbugs nu a gasit erori potentiale atunci cand a fost aplicat pe Webserver.java.

*Rezultatul complet se poate vedea in html-ul asociat.

Checkstyle:

Checkstyle verifica daca programele Java sunt scrise in concordanta cu reguli de codare specificate.

Urmatoarele stafuri au fost date:

Eficienta:

1. Sa ascund contructorul clasei utilitare Mainul.
    a. Util? → Nu
    b. Actiune: Nu se va introduce un constructorul explict privat deoarece nu are sens.

Mentenanta:

1. **Sa omit importuri de tip ".*".**
    a. Util? → Da
    b. Actiune: Sa se importeze doar ceea ce este necesar. Dupa cum se poate vedea in analiza dinamica facuta inainte de a face vreo modificare, numarul de clase incarcate este 2006, desi eu am doar doua. Numarul mare se datoreaza tocmai importarii cu ".*".
2. Complexitatea expresiei booleene:
    a. Util? → Nu chiar
    b. Actiune: Nu se va rescrie if-ul din metoda update_webserver_root_directory. Ar deveni mai scurt if-ul, dar s-ar adauga mai multe return-uri in schimb.
3. Complexitate ciclomatica:
    a. Util? → Nu chiar

    b. Actiune: Nu se va rescrie metoda handleClient. Intr-adevar are try chatch in try catch, if in if in if si un while, dar este usor de urmarit si testat in varianta in care este.
4. Separator linie goala:
    a. Util? → Nu
    b. Actiune: Nu se va introduce linii goale intre declaratii de variabile.
5. Lungimea liniei de cod:
    a. Util? → Nu
    b. Actiune: Nu se va scurta liniile de cod metionate. Ele au o lugime decenta, doar ca sunt indentate, indentare care adauga la lungimea liniei considerabil. In rest, se refera la caile absolute, cai care adauga la lungime, dar nu se considera utila intoducerea de varibile explicite.
6. Javadoc comentariu lipsa:
    a. Util? → Nu
    b. Actiune: Nu se vor intoduce explicatii direct in cod in cazul acesta. Acestea se vor scrie intr-un document separat pentru a nu incarca vizual codul.
7. Multiplii litarali de tip String:
    a. Util? → Nu
    b. Actiune: Nu se vor sterge literalii. Sunt fie cuvinte cheie, fie cai absolute, fie nume de cai standard.
8. If-uri imbricate:
    a. Util? → Nu
    b. Actiune: Nu se vor rescrie if-urile din metoda handleClient deoarece sunt intuitive si usor de urmarit.
9. O clasa top-level SI Numarul de tipuri exterioare:
    a. Util? → Nu in cazul asta.
    b. Actiune: Nu se va pune clasa Mainul intr-un alt fisier doar pentru a avea o clasa/fisier java. Asta ar fi util cand este vorba de o ierarhie mai mare de clase.
10. Declarare de pachet lipsa SI Info pachet lipsa:
    a. Util? → Nu in cazul asta
    b. Actiune: Nu se va aduga un pachet explict. Pachetul este default prin conventie si nu are nume. Nu se va adauga comentarii la pachetul default.
11. Numarul de return-uri:
    a. Util? → Nu
    b. Actiune: Nu se va modifica structura codului di metoda getFilePath. Aici se poate vedea cum rezolvarea unei probleme duce la alta. Mai devreme in 2. s-a semnat ca if-ul are complexitate ciclomatica mare. Solutia ar fi sa il sparg in mai multe sub-if-uri, dar datarita return-urilor adaugate, ar fi semnalat o problema ca aceasta in 11 sau char si 8.
12. Separator de un spatiu:
    a. Util? → Nu
    b. Actiune: Nu se va sterge spatiul.
13. Modificarea visibilitarii:
    a. Uti? → Nu in cazul acesta.
    b. Actiune: Nu se vor seta vizibilitatea metodelor pe private si nu se vor introduce metode accesoare deoarece webserver-ul creat nu este intentionat a fi lasat pe piata si exins in continuare de 3rd parties.

Fiabilitate:

1. Designul pentru extindere:
    a. Util? → Nu in cazul acesta
    b. Actiune: Nu se vor adauga comentarii deoarece vor fi adaugate intr-un document separat, si nici nu este destinat serverul pentru a fi pus pe piata si extins de 3rd parties. In plus, desi nu sunt menite a fi extinse clasele, nu se vor pune de tip final deoarece e irelevant din moment ce nu sunt preocupata de securiatea claselor -- pentru ca nu public serverul .
2. Equals omit null:
    a. Util? → In general da, dar nu in cazul de fata.
    b. Actiune: Se refera la equals-ul care se face intre cuvintele cheie “Running” si “Maintenance” cu starea serverului. Din moment ce singurul loc (momentan) unde setez starea serverului este in clasa Mainul inainte sa se intre in handleClient, nu va aparea cazul ca starea serverului sa nu fie initializata cand se executa handleClient. Eu vor urmari sa o setez corespunzator, iar cum doar eu lucrez pe cod si stiu ce fac, nu va fi setata niciodata pe null. Altfel as fi facut starea serverului private si in setter-ul corepunzator as verifica ca aceasta sa fie setata doar pe una din cele trei variante: Running, Maintenance sau Stopped.
3. Numar magic:
    a. Util? → Nu.
    b. Actiune: ‘8080’ nu este un numar magic. Poate fi orice port liber, dar fiindca acest port este de obicei liber, a fost acesta ales ca si defaut. In legarea webserverului cu GUI-ul, se va face modificari minore ca sa se poata da portul introdus prin campul corespunzator din interfata.
4. Variabile locale finale:
    a. Util? → Nu
    b. Actiune: Nu se va adauga final explicit. Variabilele locale sunt deja de acest tip prin default.
5. Parametrii finali:
    a. Util? → Nu
    b. Actiune: Nu se vor seta ca final deoarece nu aduce beneficii in contextul de fata.
6. Campuri ascunse:
    a. Util? → Nu in cazul de fata.
    b. Actiune: Nu se vor redenumi parametrii doar ca sa fie diferiti de cei ai clasei deoarece se aplica corect “this” pentru a distinge intre ei.
7. Ordinea importurilor:
    a. Util? → Nu
    b. Actiune: Nu se vor rescrie importurile ca sa satisfaca conventia de ordonare si grupare a importurilor pentru ca este irelevanta in cazul de fata.
8. Indentare:
    a. Util? → Nu
    b. Actiune: Se vor indenta astfel incat sa fie usor de urmarit.
9. Variabila javadoc:
    a. Util? → Nu
    b. Actiune: Nu se vor adauga comentarii deoarece variabilele au nume sugestive.
10. Paranteza stanga:
    a. Util → Nu
    b. Actiune: Nu se vor pune parantezele stangi pe linia de deasupra -- preferinta.

11. Numele variabilei locale SI Numele metodei:
    a. Util? → Nu
    b. Actiune: Nu se vor modifica numele.
12. Constructor lipsa:
    a. Util? → Nu in cazul de fata
    b. Actiune: Nu se va adauga un constructor la clasa Mainul.
13. Trebuie paranteze:
    a. Util? → Nu in cazul de fata.
    b. Actiune: Nu se vor adauga paranteze doarece se doareste a se executa doar linia de cod scrisa exact sub if/else-urile mentionate.
14. **Asignarea parametrilor:**
    a. Util? → Da
    b. Actiune: Se va sterge asignarea facuta, si se va pune calea direct in return.
15. Numele parametrilor:
    a. Util? → Nu
    b. Actiune: Nu se vor modifica numele.
16. Paren pad:
    a. Ultil? → Nu
    b. Actiune: Nu se vor sterge spatiile.
17. Parantezele drepta:
    a. Util? → Nu
    b. Actiune: Nu se vor pune pe linia de deasupra corespunzatoare.
18. Comentariu scris dupa linia de cod:
    a. Ultil? → Nu
    b. Actiune: Nu se vor sterge comentariile.
19. Main necomentat:
    a. Ultil? → Nu
    b. Actiune: Nu se va adauga comentariul in cod, ci in documentatie.
20. Spatiu dupa:
    a. Ultil? → Nu
    b. Actiune: Nu se vor sterge spatiile.
21. Spatii in jur:
    a. Ultil? → Nu
    b. Actiune: Nu se vor sterge spatiile.

Analiza dinamica

Analiza dinamica a fost facuta cu VisualVM. VisualVM este un tool care ofera o interfata vizuala care ofera informatii detaliate despre aplicatii Java in timp ce acestea ruleaza pe o masina virtuala Java.

Inainte si dupa a face modificarile care au fost decise in urma analizei statice, analiza dinamica a oferit urmatoarele rezultate:

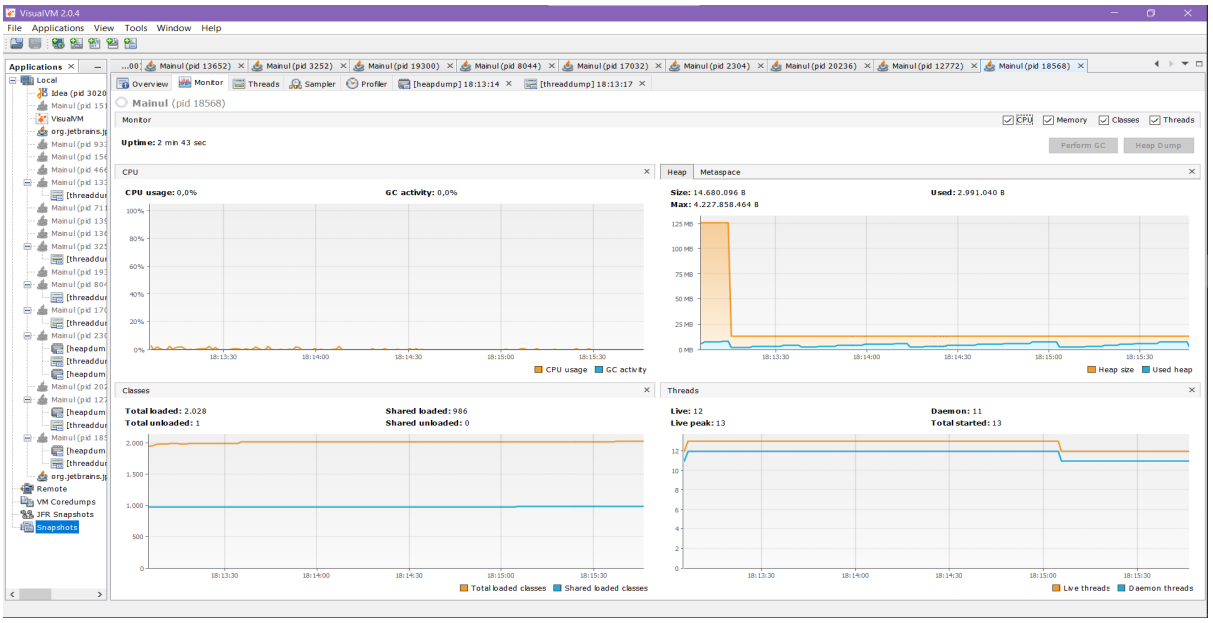

Fig 1.

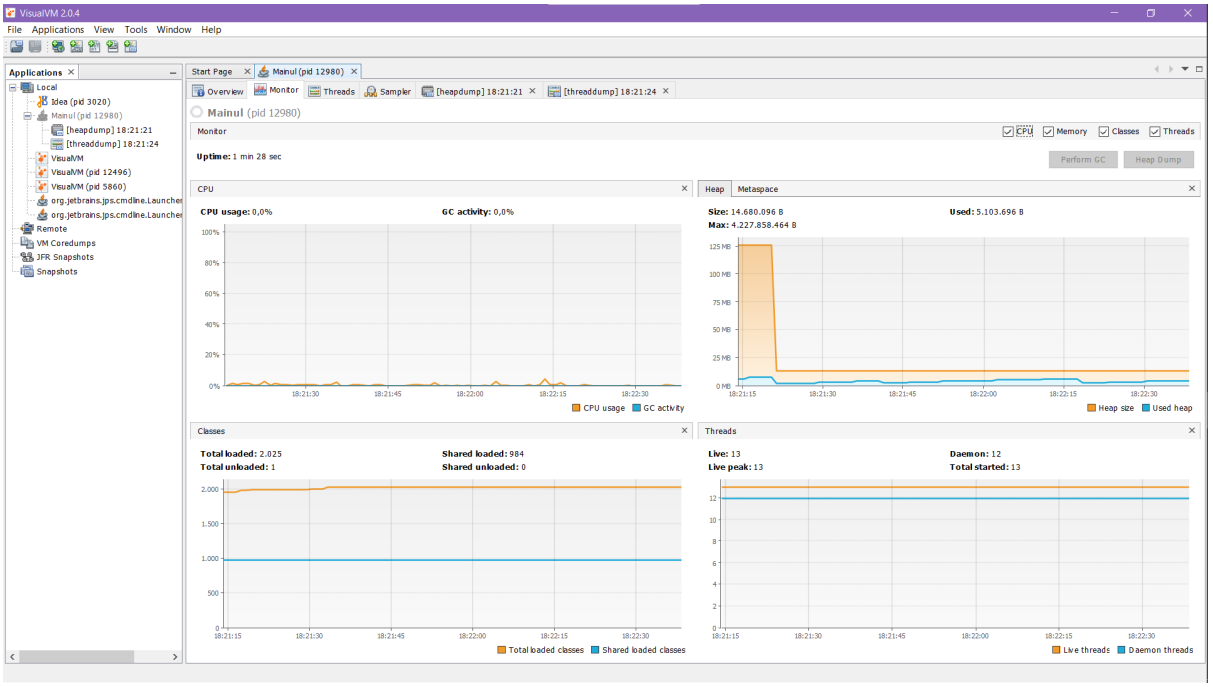

Fig 2.

Ce s-a intamplat?

Cu o trei clase incarcate mai putin:
Inainte de a adauga doar importurile strict necesare, 2028 clase au fost incarcate. Dar dupa ce au fost adaugate importurile specific, doar 2025 au fost incarcate. Nu e o diferenta remarcabila in cazul de fata, desi in general poate face o diferenta mai importanta si prin urmare sa fie justificata importarea stricta. Motivul pentru care diferenta nu a fost majora, este ca poate clasele importate sunt de baza/core. Asta inseamna ca sunt destul de sus in ierarhie si indiferent de cum au fost importate - cu * sau specific -, cam tot aceleasi clase vor trebui incarcate tocmai pentru motivul ca acestea sunt de baza.

Ce thread-uri am creat noi?

1. Main -- Singurul non-Daemon
2. TCP-Accept -- Daemon
3. TCP-Connection 1 -- Daemon
4. TCP-Connection2 -- Daemon

Cand si de ce au fost create?

Main:
1. Cand? → Cand a inceput executia programului. Tot timpul exista un thread Main
2. De ce? → Fara main nu se executa codul scris.

TCP-Accept:
1. Cand? → Cand s-a executat handle_one_request(serversocket.accept()) apelata din Main
2. De ce? → Serverul “asculta” cereri de conectare ca sa le poata accepta. Pentru a putea asculta incontinuu pana ce se efectueaza o cerere, un thread trebuie pornit ca sa faca asta.

TCP-Connection1:
1. Cand? → Cand s-a executat handleClient(clientSocket.getOutputStream()) apelata din handle_one_request.
2. De ce? → Pentru a mentine un canal de comunicare deshis de la client la server.

TCP-Connection2:
1. Cand? → Cand s-a executat handleClient(BufferedReader(new InputStreamReader(clientSocket.getInputStream()))) apelata din handle_one_request.
2. De ce? → Pentru a mentine un canal de comunicare deshis de la server la client.

Asa cum se vede din ThreadDump-ul atasat, toate threadurile de interes sunt RUNNABLE. Motivul este ca, in timp ce main ruleaza, toate celelalte thread-uri de interes asteapta sa se intample ceva. Adica asteapta ca un client sa se conecteze ca sa i se poata da accept si ca sa se transmita ceva pe canalele de communicare de la/inspre client. Cum, atunci cand apasam pe butonul care afiseaza thread-dump-ul, nici un client nu incearca sa se conecteze (pentru ca avem conexiuni non-persistente si noi le initializam prin navigarea pe site) si cum nu se transmite

nimic pe canalele de comunicare, toate thread-urile de interes sunt afisate ca fiind in asteptarea celor mai de sus sa se intample -- ceea ce si sunt, pentru ca eu pot face doar un lucru de-o data, ori navighez pe site, ori cer afisarea thread-dump-ului.

Consumul de memorie

Consumul de memorie cand navigam pe site este urmatorul:

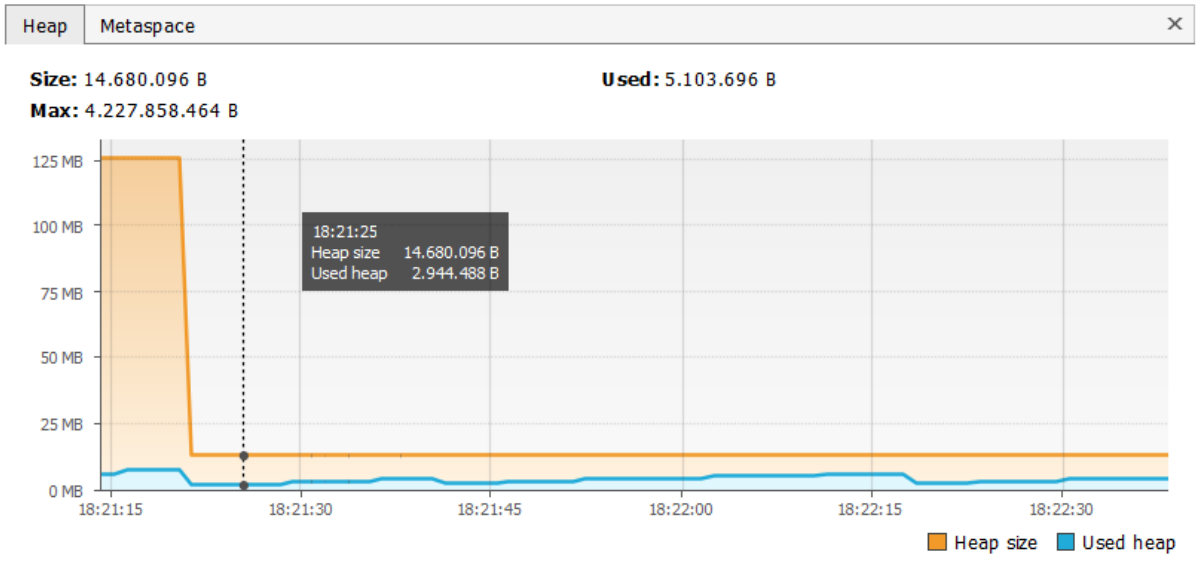

Fig 3.

E modest, dar tinand cont ca serverul e destul de minimal, consumul e rezonabil.

Starea thread-urilor

Thread-urile de interes sunt RUNNING.

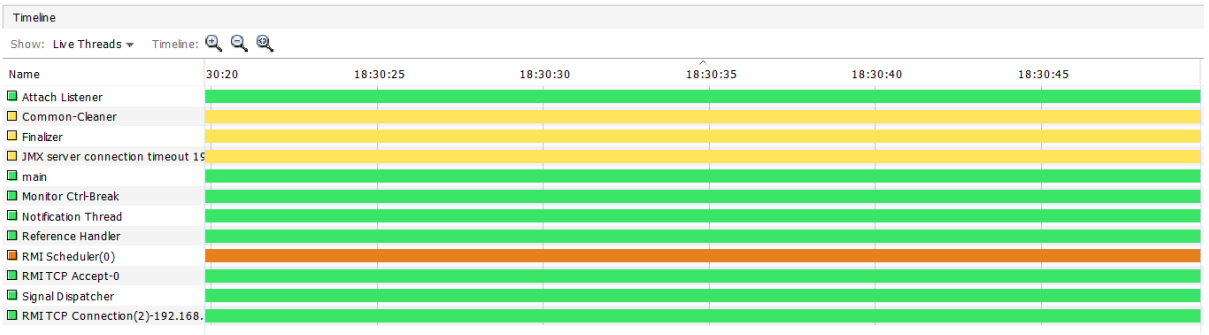

Fig 4.

Si la fel sunt si in thread-dump - care poate fi gasit intr-un fisier separat.

Heap dump

Inainte si dupa a face modificarile care au fost decise in urma analizei statice, analiza dinamica a oferit urmatoarele rezultate:

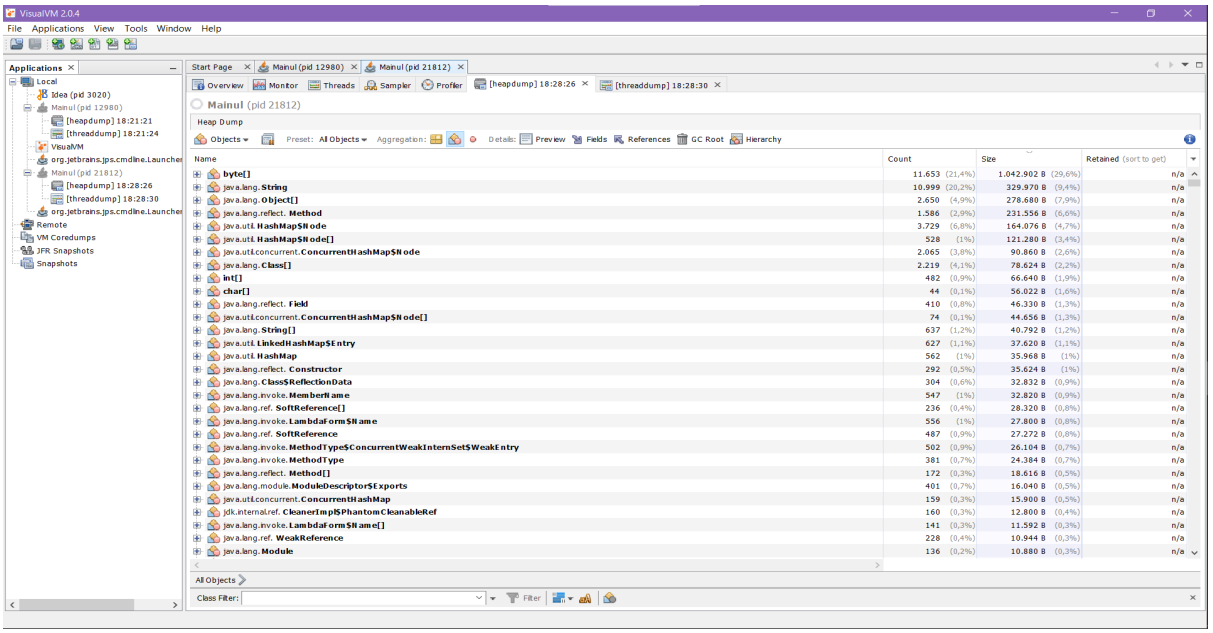

Fig 5.

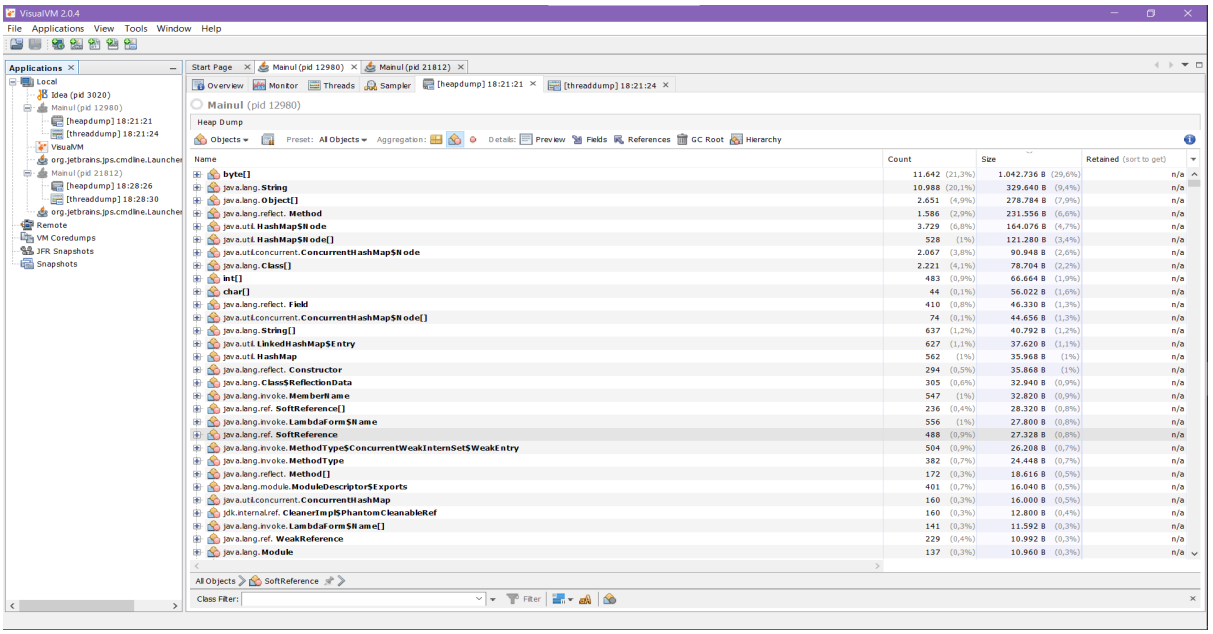

Fig 6.

Se observa ca numarul de instante byte[] si String descreste de la o poza la alta - lucru care era de asteptat. Cand se da o privire mai atenta la String-urile salvate, se poate observa ca String-urile sunt salvate in byte-uri - de aceea exista o corelatie intre cele doua. De asemenea, si numarul de metode si obiecte a scazut, iar acest lucru se datoreaza tot numarului mai mic de clase incarcate dupa modificarile ce au fost facute.