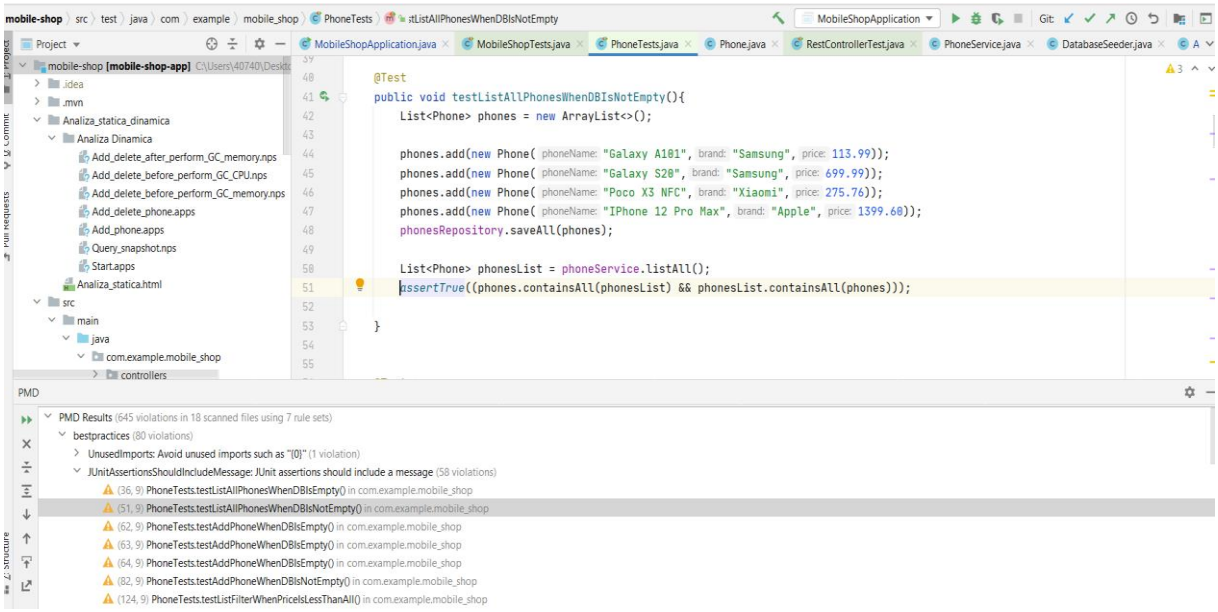
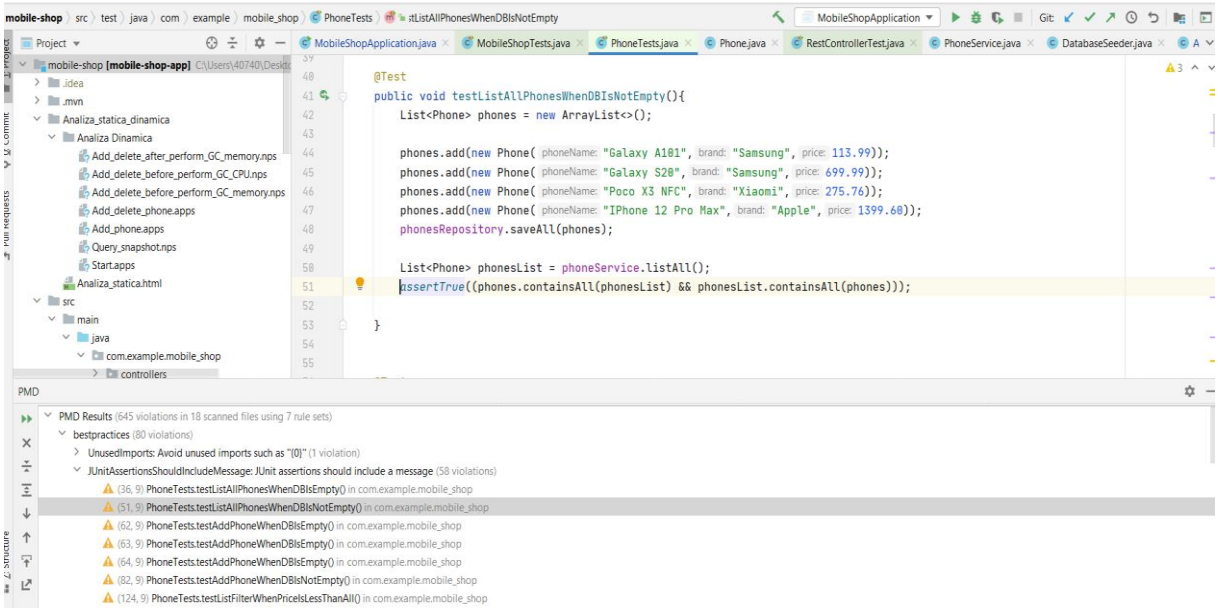
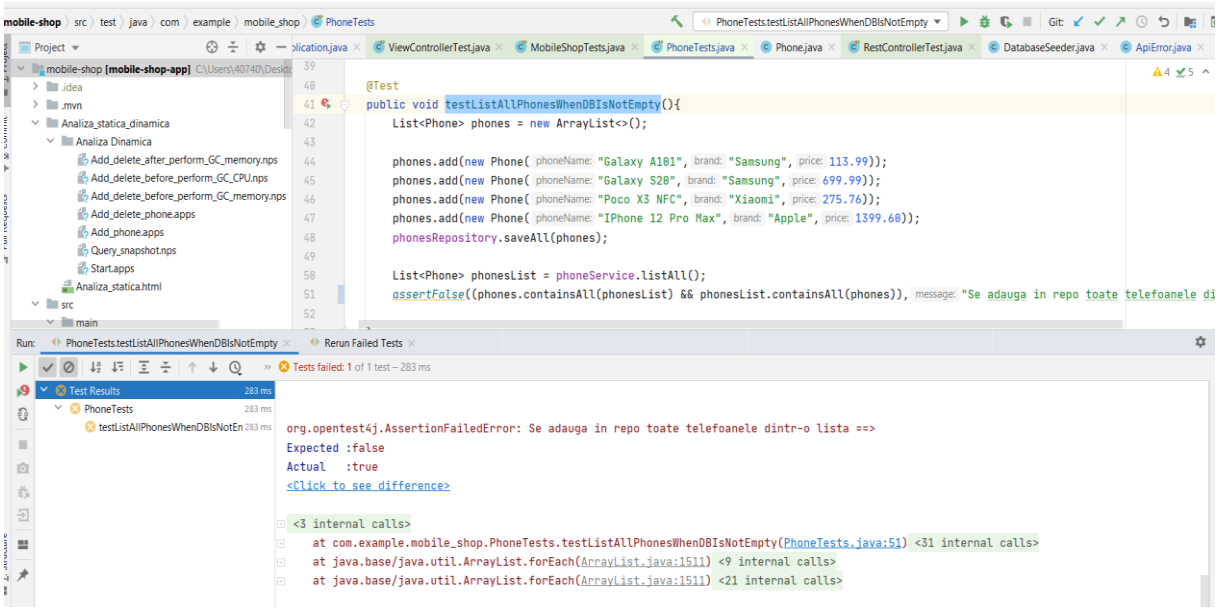
Analiza statica – cu PMD

- best practices – paranteze nefolositoare -> nu trebuie puse si la conditie, numai la assert

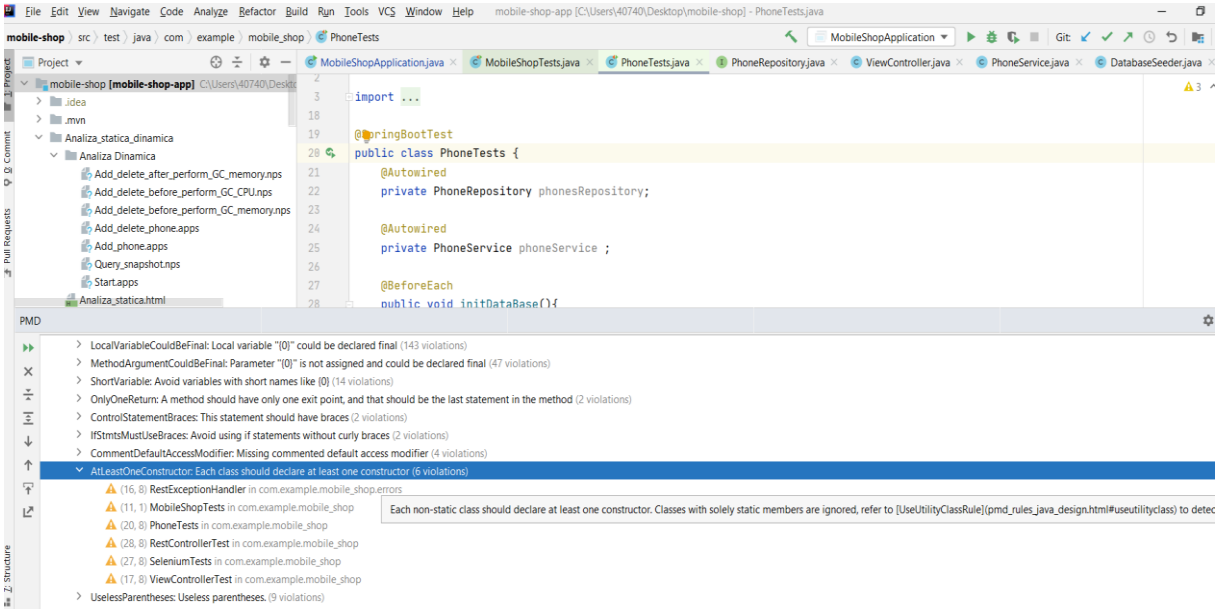


- best practices – assert-urile ar trebui sa contina un mesaj cu ce anume s-a testat pentru a identifica mai bine greseala in caz ca testul fail-uie



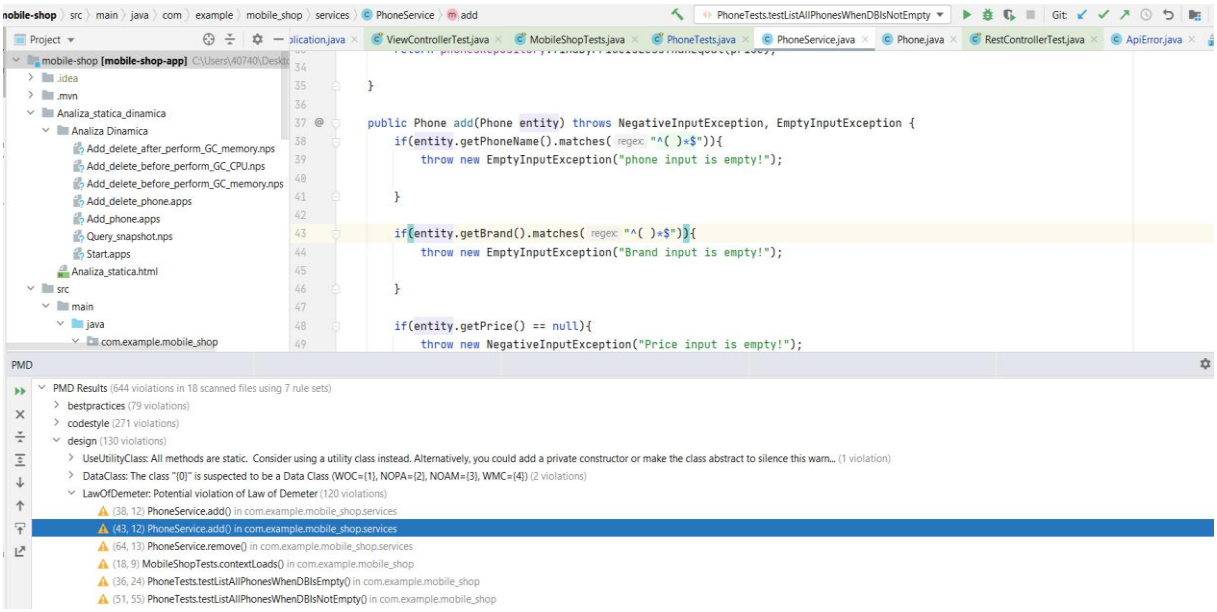


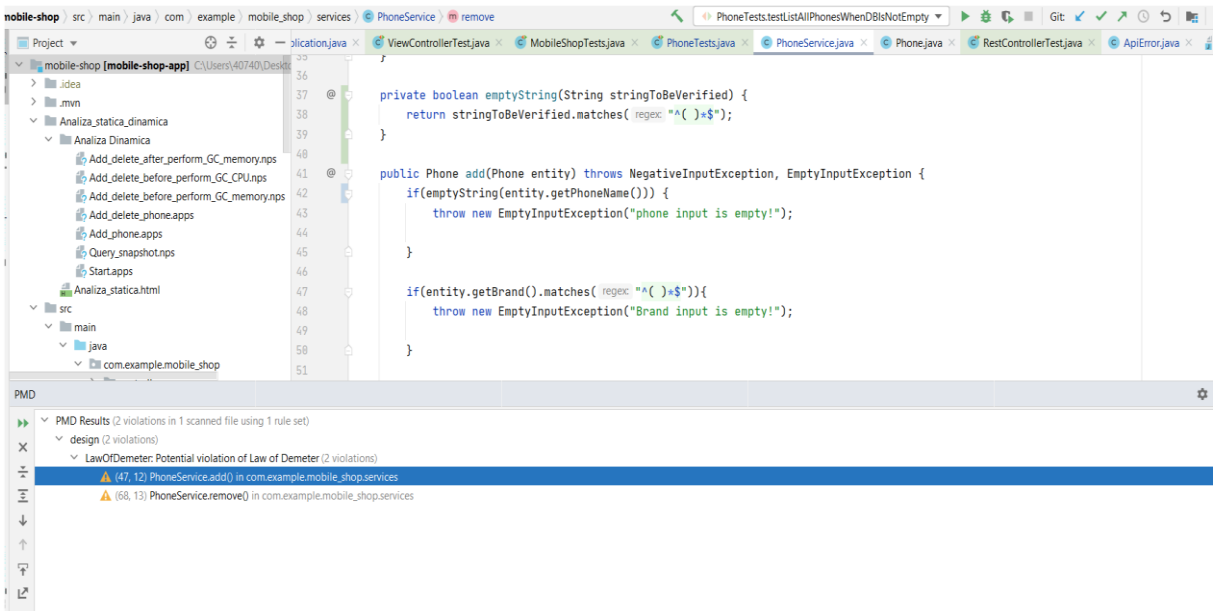
- codestyle – fiecare clasa ar trebui sa contina cel putin un constructor, dar clasele de test se presupune ca nu au nevoie sa fie instantiate, prin urmare nu consider acest lucru un bug



- design – “incalcarea” legii lui Demeter, prin apelarea metodei matches pe un obiect al argumentului entity, si anume string-ul phoneName

- se rezolva prin adaugarea metodei emptyString care ia ca parametru phoneName si apeleaza pe el matches pentru a nu se face apelul pe phoneName returnat de un getter





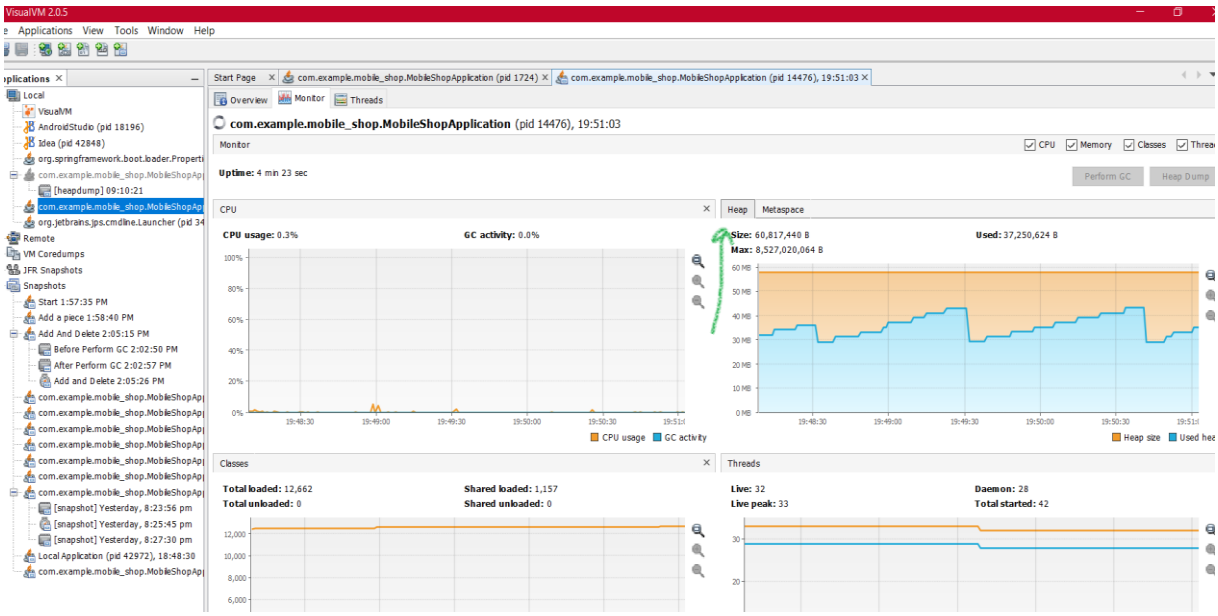
Analiza dinamica - cu Visual VM

In cazul de fata, nu ar exista probleme vizibile in ceea ce priveste consumul de memorie(HEAP, unde se stocheaza obiectele create) si timpul de raspuns al request-urilor.

Exemple de probleme:

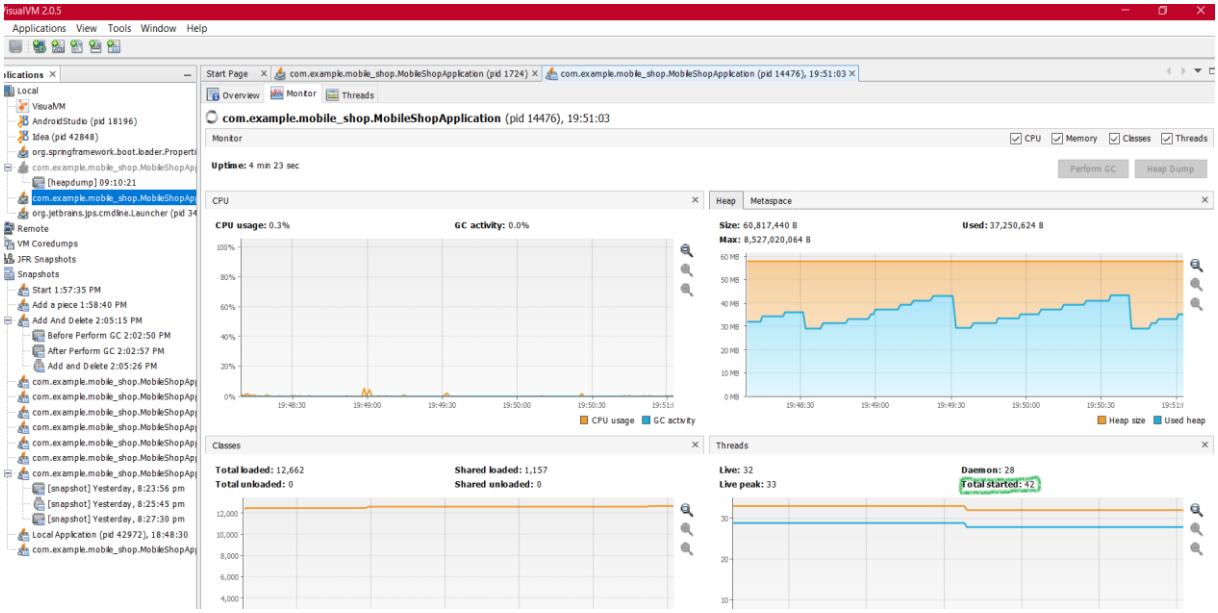
Un caz de incarcare a HEAP-ului ar fi creerea unor obiecte de dimensiuni mari, ceea ce nu este cazul pentru ca obiectele de tip telefon au ca si proprietati doar string-uri si un int, dar daca ar avea in componenta obiecte de mari dimensiuni ar ocupa multa memorie.

Deasemenea, foarte multe creari de obiecte noi ar incarca memoria. Utilizarea memoriei HEAP se poate observa aici:



O solutie ar fi configurarea dimensiunii HEAP-ului, prin setarea unor parametrii pentru JVM. De asemenea, se poate configura JVM-ul pentru optimizarea GC, in cazul in care se fac multe stergeri de exemplu. O cantitate mare de obiecte “de curatat” , poate determina GC sa creasca latenta si sa foloseasca mai multe resurse. Se poate seta, de exemplu, ca GC sa ruleze concurent cu procesele aplicatiei si/sau sa “curete” cele mai recente obiecte din memorie.

Avand in vedere ca se foloseste serverul Tomcat, numarul de thread-uri care ruleaza in paralel este echivalent cu numarul de request-uri http facute catre server, in acelasi timp. Conectorul HTTP default se blocheaza, existand legatura 1 conexiune-1 thread-1 request. In cazul in care exista foarte multe thread-uri, se pot irosi resurse, pentru ca o conexiune nu e neaparat folosita la maximum. Cum nu am implementat posibilitatea crearii mai multor conturi pentru trimitere de request-uri de la mai multi useri si nici nu am generat mai multe request-uri concurente, cresterea semnificativa a numarului de thread-uri nu e vizibila in Visual VM la sectiunea Threads. Acest lucru s-ar observa aici:



O solutie ar fi configurarea serverului Tomcat pentru a folosi un conector NIO non-blocking, care pastreaza conexiunea activa pentru toate thread-urile create de request-uri. Dar, in ceea ce priveste timpul de executie, metoda blocking este mai rapida, dar daca exista posibilitatea mai multor conexiuni idle, e preferabila folosirea conectorului NIO.