**Cerințe obligatorii**

1. Pattern-urile implementate trebuie sa respecte definitia din GoF discutată în cadrul cursurilor și laboratoarelor. Nu sunt acceptate variații sau implementării incomplete.
2. Pattern-ul trebuie implementat corect în totalitate corect pentru a fi luat în calcul
3. Soluția nu conține erori de compilare
4. Testele unitare sunt considerate corecte doar dacă sunt implementate conform cerințelor și dacă metodele sunt corectate corespunzător pe baza lor
5. Pattern-urile pot fi tratate distinct sau pot fi implementate pe același set de clase

**Cerințe Clean Code obligatorii (soluția este depunctata cu câte 5 puncte pentru fiecare cerința ce nu este respectată**) - maxim se pot pierde 15 puncte

1. Pentru denumirea claselor, funcțiilor, testelor unitare, atributelor și a variabilelor se respecta convenția de nume de tip Java Mix CamelCase;
2. Pattern-urile, Test Case-urile, Excepțiile și clasa ce contine metoda main() sunt definite in pachete distincte ce au forma *cts.nume.prenume.gNrGrupa.teste*, *cts.nume.prenume.gNrGrupa.patternX*, *cts.nume.prenume.gNrGrupa.main* (studenții din anul suplimentar trec “as” în loc de gNrGrupa)
3. Clasele și metodele sunt implementate respectând principiile KISS, DRY și SOLID (atenție la DIP)

Se dezvoltă o aplicație software ce reprezintă un plugin antivirus pentru un browser Web.

**10p.** Dezvolți o soluție software/plugin care sa permită verificarea fișierelor descărcate de pe Internet și a link-urilor Web accesate de utilizatori prin intermediul browser-ului. Această soluție este bazată pe module ce vor verifica acțiunile utilizatorilor, module ce au interfața ***IAntivirus*** (enumerarea se poate muta într-o clasa separată). Modulele vor face o verificare completă a acțiunii utilizatorului conform următoarelor condiții:

* dacă fișierul descărcat este de tip .exe (dacă numele fișierului conține .exe) se verifică dacă conține viruși; dacă da atunci download-ul este întrerupt (se afișează mesaj), altfel se trece la următoarea verificare;
* dacă fișierul descărcat este de tip .pdf se verifică dacă conține malware; dacă da atunci download-ul este întrerupt (se afișează mesaj), altfel se trece la următoarea verificare;
* dacă se accesează un link se verifică dacă este unul de tip https; dacă nu este atunci este alertat utilizatorul printr-un mesaj; în ambele situații link-ul este accesat;
* ultimul modul va accesa link-ul sau va descarcă fișierul și va consemna această acțiune în istoricul acțiunilor (istoricul se poate simula printr-o colecție standard ce este gestionată de acest modul)

Modulele pot avea propriile lor atribute (cum este cazul modulului ce gestionează istoricul).

**5p.** Pattern-ul este testat în main() prin simularea descărcării unui fișier .exe ce conține un virus (de ex. virus.exe), a unuia care nu conține viruși (safeExam.exe), a unor fișiere .pdf cu sau fără malware și a unor url-uri cu sau fără https. Toate acțiunile duse la bun sfârșit trebuie să se regăsească în istoric.

**5p.** Independent de soluția anterioară să se găsească o soluție prin care un modul al plugin-ului să poată fi modificat/customizat în timp real prin adăugarea de setări noi. Pornind de la un exemplu de modul ce implementează interfața ***IAntivirus*** și care conține minim 2 atribute la alegere (de ex. se pornește de la modulul care verifica fișiere, sau cel care verifică link-uri), găsiți o soluție care să permită utilizatorului să îl modifice în timpul execuției plugin-ului prin adăugarea a cel puțin o setare nouă. Această setare va afecta și modul în care funcționează verificarea acțiunii utilizatorilor.

**5p.** Să se testeze soluția prin definirea modului inițial si procesarea a minim 2 acțiuni. Se va modifica modulul prin adăugarea setării suplimentare și se vor procesa alte minim 2 acțiuni.

**6p.** Dându-se clasa *TestGrila* și restricțiile definite prin comentarii să se implementeze teste unitare (toate testele unitare aferente unei metode sunt implementate într-un TestCase) care să cuprindă:

1. un unit test care să realizeze o testare *Boundary* pentru **setPunctajMaxim()** **(1.5p)**
2. un unit test care să testeze o testare *Error Condition* pentru **setPunctajMaxim() (1.5p)**
3. un unit test de tip *Existence* pentru metoda ***calculTotalPuncte()* (1.5p)**
4. un unit test de verificare de tip *Cardinality = 2* pentru metoda ***calculTotalPuncte*()**;**(1.5p)**

**2p.** Să se implementeze o suită de teste care să conțină DOAR câte o metodă, la alegere, din fiecare test case

**2p.** Prin testele implementate sau prin adăugarea de teste noi sa se testeze *calculTotalPuncte*()asigurând un code coverage de 100% pentru această metodă.