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 conventia 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 destinată unui serviciu de Email.

10p. Soluția trebuie să permită gestiunea pe categorii a clienților de email. Clienții email trebuie să fie capabili să deruleze acțiuni specifice interfeței date dar și să permită gestiunea de grupuri de email compuse din alte entități. Entitățile din soluție conțin și cel puțin 3 atribute (cu tipuri diferite) specifice unui client de Email.

5p. Pattern-ul este testat în main() prin definirea și utilizarea unei structuri ierarhice care să conțină minim 3 niveluri cu minim 6 elemente. Pe nivelurile 2 și 3 trebuie să existe cel puțin un utilizator de email și un grup de email. Pentru ierarhie sau pentru anumite elemente din ierarhie se vor apela metodele specifice interfeței (dacă implementarea permite apelul lor doar pentru entități sau pentru entități și grupuri).

5p. Anumiți clienți de email care implementează interfața primită trebuie modificați în timpul execuției (la run-time). Soluția propusă trebuie să permită modificarea unor entități prin adaugarea a cel puțin un atribut nou, ce este prelucrat în una dintre funcțiile din interfață. De asemenea, pentru anumite obiecte trebuie să fie posibilă modificarea uneia dintre funcțiile specifice interfeței (la alegere).

5p. Să se testeze soluția prin definirea a 4 entități distincte. Una dintre entități este modificata la run-time prin adaugarea atributului nou. Pentru o alta entitate este modificat comportamentul funcției alese. Pentru o alta entitate sunt aplicate în cascadă cele 2 modificări posibile. Pentru cele 4 entități se vor apela metodele interfeței pentru a demonstra comportamentul diferit al acestora.

6p. Dându-se clasa *Facturare,* clasa *Produs* și următoarele restricții: o factură conține maxim 20 de produse, prețul unui produs este cuprins în intervalul [1,1000] să se implementeze teste unitare, gestionate în test case-uri diferite pentru fiecare metodă testată, care să cuprindă:

1. un unit test care să realizeze o testare *Right* pentru **setPret() (1.5p)**
2. un unit test care să testeze o testare *Boundary* pentru **setPret() (1.5p)**
3. un unit test de tip *Existence* pentru metoda ***calculValoareTVA();*** dacă este nevoie de excepție se generează una de tip *ExceptieFacturaFaraProduse* **(1.5p)**
4. un unit test de verificare de tip *Inverse* pentru metoda ***calculValoareTVA*****();(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 adaugarea de teste noi sa se testeze *setPret*() din clasa *Produs* asigurând un code coverage de 100% pentru această metodă.