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 în 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 necesară gestiunii examenelor dintr-o sesiune.

10p. Soluția trebuie să permită salvarea tuturor examenelor pentru toți studenții unei facultăți. Un student susține mai multe examene, câte un examen pentru fiecare disciplină. Fiecare disciplină are un număr diferit de credite. Să se implementeze soluția ținând cont de optimizarea memoriei utilizate pentru salvarea tuturor examenelor pentru toți studenții facultății.

5p. Pattern-ul este testat în main() prin definirea a cel puțin trei studenți care au susținut examene la cel puțin două discipline diferite.

5p. Pentru susținerea examenului există posibilitatea ca acesta să fie sub formă practică, sub formă orală sau sub formă unui test grilă. Să se implementeze modulul care permite stabilirea modului de susținere a examenului pentru un Student (Interfața IStudent) în momentul examenului.

5p. Să se testeze soluția prin susținerea a cel puțin trei examene în moduri diferite.

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 *CrossCheck* 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.