**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 destinată unei platforme de livrare produse.

**10p.** Dezvolți o soluție software care sa permită clienților sa comande produse alimentare prin intermediul unor servicii similare Glovo/FoodPanda. Clienții comanda produsele respective urmând ca aceasta sa fie preluată de un curier disponibil în sistem. Comanda va fi livrată în viitor însă timpul de livrare depinde de mai mulți factori externi (trafic, aglomerația din magazin, etc). Comenzile înregistrate în sistem nu mai pot fi anulate, nu sunt pierdute si vor fi onorate. Comenzile implementează interfața ***IGlovo*** si conțin numele magazinului preferat și încă un detaliu la alegere (care sa aibă sens). Curierii vor face livrarea comenzii prin acțiunea *livrareComanda*() si când acest lucru se întâmpla se va afișa numele clientului in sistem. Curierii sunt descriși prin nume si un rating cu valori intre 1 si 5.

**5p.** Pattern-ul este testat în main() prin lansarea a minim 3 comenzi diferite (la magazine diferite si pentru clienți diferiți). Pentru cele 3 comenzi se folosesc cel puțin 2 curieri diferiți. Comenzile pot fi stocate într-o colecție standard direct in main() sau prin intermediul unei clase container separata. Comenzile vor fi executate in ordinea in care au fost înregistrate si vor fi scoase din colecție. La execuția comenzilor se va afișa in consola un mesaj ce conține numele clientului (vezi cerința anterioara privind livrarea)

**5p.** Se considera ca în acest moment fiecare client poate include în comanda orice fel de produse însă soluția trebuie sa permită pe viitor adăugarea unor filtre/reguli (nu permite comenzi mai mici de o anumită sumă, nu permite comenzi pentru anumite produse, adaugă costul transportului la comanda, etc) care sa limiteze/controleze opțiunile clienților. Aceste modificări trebuie incluse în serviciile din back-end și trebuie sa controleze modul în care se realizează o comanda. Separat de cerința anterioara se considera ca in acest moment comenzile sunt procesate in backend de un modul ce implementează interfața ***IProcesareComenzi***. Sa se adauge un modul nou care sa filtreze comenzile după o regula la alegere sau care sa adauge costul transportului (considerat suma fixa). După acest pas, comenzile ok sunt transmise modulului existent.

**5p.** Să se testeze soluția prin definirea modului inițial si procesarea a minim 2 comenzi. Sa se adauge modulul nou in sistem si sa se exemplifice filtrarea/procesarea suplimentara a comenzilor (minim 2 comenzi).

**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.