**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 Trading.

**10p.** Dezvolți o soluție software de tip Trading care sa permită clienților să cumpere acțiuni pe diferite burse prin intermediul unor brokeri. Clienții pot comanda cumpărarea sau vânzarea unui număr de acțiuni pentru companii listate la bursa conform interfeței ***IClientBursa***. Clienții își aleg brokerul în momentul în care lansează comanda în sistem. Comenzile sunt înregistrate în sistem și vor fi procesate în ordinea în care au fost generate de platforma de Trading, în funcție de gradul de încărcare al platformei. Fiecare comanda conține numele clientului, starea ei (procesata sau nu), tipul ei (pentru vânzare sau cumpărare) si încă un atribut la alegere (orice are sens). Brokerul va procesa orice comanda, indiferent de tipul ei, printr-o singura acțiune (se poate denumi oricum).

**5p.** Pattern-ul este testat în main() prin lansarea a minim 4 comenzi diferite (cumpărare, vânzare pentru diferite companii). 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.

**5p.** În timp platforma se va extinde si vor fi oferite diferite tipuri de servicii financiare: schimb valutar, vânzare sau cumpărare de crypto-monede, investiții in fonduri, cumpărare de bonuri de trezorerie. Aceasta dezvoltare trebuie susținuta de o soluție care sa permită extinderea tipurilor de servicii. Toate aceste servicii expun interfața *IFinanciar* ce conține 2 metode diferite (2 metode la alegere cu număr diferit de parametrii)

**5p.** Să se testeze soluția prin definirea a 2-3 tipuri diferite. Pentru fiecare tip se va construi un obiect si va fi apelata una dintre metodele interfeței..

**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 *Right* pentru **setDificultate()** **(1.5p)**
2. un unit test care să testeze o testare *Error Condition* pentru **setPunctajMaxim() (1.5p)**
3. un unit test de tip *Ordering* pentru metoda ***calculTotalPuncte()* (1.5p)**
4. un unit test de verificare de tip *Cardinality = 1* 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 *setDificultate* ()asigurând un code coverage de 100% pentru această metodă.