**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)
4. Denumirile de clase, metode si atribute au semnificație (au legatura cu subiectul) în contextul subiectului. NU se folosesc denumiri generice/teoretice

Se dezvoltă o aplicație software destinată administrării unui cartier de blocuri.

**10p.** Dezvolți o soluție software pentru asociațiile de proprietari dintr-un cartier. Fiecare bloc are locatari, are un administrator și fiecare scară de bloc are un președinte de scară. Președinții de scară se ocupă doar de problemele locatarilor de pe scara sa, iar administratorii gestionează toate scările unui bloc. Administratorii la rândul lor vor raporta unui reprezentant al primăriei aceste probleme. Să se implementeze modulul care permite realizarea unei structuri pentru reprezentarea ierarhică a **locatarilor, președinților de scara și a administratorilor de blo**c. Soluția este dezvoltată în jurul clasei abstracte *LocatarAbstract* ce nu poate fi modificata.

**5p.** Pattern-ul este testat în main() prin exemplificarea scenariului în care avem un bloc cu minim 2 scări, fiecare scara avand minim 3 locatari. Structura trebuie sa contina și administratorul blocului.

**5p.** În cadrul soluției un administrator trebuie să poată anunța toți locatarii (utilizați interfața ***ILocatar***) din bloc atunci când afișează la avizier costurile cu întreținerea. Să se realizeze un modul care să permită administratorului să gestioneze eficient o listă de locatari pe care să-i anunțe automat atunci când face acest lucru.

**5p.** Să se testeze soluția prin definirea a minimum doi administratori care vor publica la avizier întreținerea și anunță locatarii (minim 2 pentru fiecare bloc).

**6p.** Dându-se clasa *Avion* 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 **setGreutateIncarcata()** **(1.5p)**
2. un unit test care să testeze o testare *Error Condition* pentru **setGreutateIncarcata().** Pentru test se va folosi o excepție custom **(1.5p)**
3. un unit test de tip *Existence* pentru ***constructorul cu parametri* (1.5p)**
4. un unit test de verificare de tip *Range* pentru ***constructorul cu parametri***;**(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 *getPretBilet*() asigurând un code coverage de 100% pentru această metodă.