Cerințe obligatorii

1. Pattern-urile implementate trebuie sa respecte definiția din GoF discutată în cadrul cursurilor și laboratoarelor. Nu sunt acceptate variații sau implementări incomplete.
2. Pattern-ul trebuie implementat corect în totalitate pentru a fi luat în calcul.
3. Soluția nu conține erori de compilare.
4. Pattern-urile pot fi tratate distinct sau pot fi implementate pe același set de clase.
5. Implementările care nu au legătura funcțională cu cerințele din subiect NU vor fi luate în calcul (preluare unui exemplu din alte surse nu va fi punctată).
6. NU este permisă modificare claselor/interfetelor primite.
7. Soluțiile vor fi verificate încrucișat folosind MOSS. Nu este permisă partajarea de cod între studenți. Soluțiile care au un grad de similitudine mai mare de 30% vor fi anulate.

Cerințe Clean Code obligatorii (soluția este depunctata cu câte 2 puncte pentru fiecare cerința ce nu este respectată) - maxim se pot pierde 4 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 și clasa ce conține metoda main() sunt definite in pachete distincte ce au forma *cts.nume.prenume.gGrupa.pattern.model*, *cts.nume.prenume.Grupa.pattern.main* (studenții din anul suplimentar trec “as” în loc de gGrupa.
3. Clasele și metodele sunt implementate respectând principiile KISS, DRY și SOLID (atenție la DIP)
4. Denumirile de clase, metode, variabile, precum și mesajele afișate la consola trebuie sa aibă legătura cu subiectul primit (nu sunt acceptate denumiri generice). Funcțional, metodele vor afișa mesaje la consola care sa simuleze acțiunea cerută sau vor implementa prelucrări simple.

Se dezvoltă o aplicație software destinată unui service auto.

4p. În cadrul aplicației de gestiune a activității unui service auto, se dorește implementarea unui modul de organizare a intrării mașinilor în service. Se dorește ca modulul să nu permită crearea mai multor service-uri în aplicație astfel încât, la un moment, să fie acceptată o singură mașină în service pentru reparații. Service-ul trebuie să implementeze interfața IService iar mașinile acceptate în service trebuie să extindă clasa abstractă AMasina.

1p. Să se testeze soluția prin crearea unui service care să gestioneze un flux de minim 4 mașini care se prezintă pentru reparații.

3p. În cadrul service-ului, mașinile sunt evaluate în funcție de tipul acesteia (SUV, VAN, SEDAN, etc). Din acest motiv este nevoie de un modul care să creeze obiecte din familia de clase ale clasei abstracte AMasina. Pentru fiecare tip de masina este folosită o clasă aferentă care moștenește clasa AMasina. Să se implementeze modulul care îi va ajuta pe angajații service-ului în procesul de creare al obiectelor din familia masinilor.

1p. Să se testeze soluția prin crearea a cel puțin patru obiecte de tip AMasina din cel puțin două tipuri diferite din familia masinilor.

Timp de lucru: 50 minute.