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 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 necesară automatizări căilor ferate.

10p. Soluția trebuie să permită implementarea unei bariere autonome care sa trateze diferit situațiile specifice unei treceri peste o cale ferată. Bariera nu este controlata din exterior. În funcție de starea unor senzori bariera implementează acțiuni specifice interfeței *ITrecereCaleFerata*. Bariera este în mod normal ridicată și aceasta pornește avertizarea sonoră în momentul în care este identificată apropierea trenului. După ce trec 10 secunde bariera se coboară automat. De asemenea, mesajele afișate de barieră diferă în funcție de starea în care se găsește bariera.

5p. Pattern-ul este testat în main() prin definirea unei bariere inteligente. Exemplificați trecerea barierei prin 2-3 scenarii diferite. Bariera fiind inteligentă exemplificați comportamentul diferit în funcție de faza în care se găsește.

5p. Pentru a evita situațiile în care bariera nu este ridicată după ce trenul a trecut sau este ridicată deși trenul este încă în zona se dorește completarea sistemului cu un senzor extern care sa facă verificări suplimentare. Senzorul existent va notifica noul senzor în loc sa transmită informația direct către bariera însă soluția propusă nu trebuie să presupună modificarea sistemului actual. Găsiți o soluție care să permită adăugarea noii componente dar menținând soluția actuală în utilizare.

5p. Să se testeze soluția prin simularea scenariului inițial (fără senzorul suplimentar). După aceasta adăugați senzorul extern si arătați că scenariul inițial nu suferă modificări.

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 *Range* 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.