

FACULTATEA CALCULATOARE, INFORMATICA SI MICROELECTRONICA
UNIVERSITATEA TEHNICA A MOLDOVEI

ANALIZA, PROIECTAREA ȘI PROGRAMAREA ORIENTATĂ PE
OBIECTE

LUCRAREA DE LABORATOR#3

Implementare principiilor SOLID Dependency Inversion și Liskov Substitution

Autor:

Marina JECHIU

lector asistent:

Mihail PECARI

Laboratory work #2

1 Scopul lucrării de laborator

Scopul lucrării de laborator este de a implementa într-un limbaj de programare principiile Dependency Inversion și Liskov Substitution.

2 Sarcina de lucru

În cadrul laboratorului e nevoie de realizat un software care o să conțină realizarea principiilor L (Liskov Substitution) și D (Dependency Inversion). Tematica este la discreția dumneavoastră (Nu se accepta folosirea lucrării precedente). Source code-ul este obligatoriu de încărcat pe github. La susținerea laboratorului prezenta raportului e obligatorie.

3 Noțiuni teoretice

Liskov Substitution Acest concept a fost introdus de Barbara Liskov în 1987 la conferința “Object Oriented Programming Systems Languages and Application”.

Barbara a declarat următoarea frază: “Dacă pentru fiecare obiect o_1 al tipului S există un obiect o_2 al tipului T astfel încât pentru orice program P comportamentul lui P este neschimbat când o_1 înlocuiește o_2 atunci S este un subtip al lui T ”. [9] Acest principiu este o extensie a Principiului Open Closed, de aceea pentru a înțelege comportamentul trebuie să fim siguri că noile clase derivate extind clasele de bază fără a le schimba comportamentul. Clasele derivate noi ar trebui să înlocuiască clasele de bază fără a produce vreo schimbare în cod.

Pentru ca o clasă derivată să poată înlocui clasa ei de bază (să se respecte LSP) trebuie să se respecte proprietățile:

- Precondițiile să nu fie mai puternice decât în metoda clasei de bază.
- Postcondițiile să nu fie mai slabe decât în metoda din clasa de bază.
- Invarianta din supertip trebuie să se conserve și în subtip. Cu alte cuvinte, metodele derivate ar trebui să nu aștepte nici mai mult nici mai puțin.

Interface segregation principle Atunci când proiectăm o aplicație trebuie să ținem cont de cum vom trata o clasă/ un modul abstract care conține și alte submodule. Dacă vrem să extindem aplicația noastră adăugând un nou modul care nu conține toate submodulele clasei inițiale, atunci suntem obligați să implementăm toate funcționalitățile interfeței și să scriem niste metode nefolositoare. O astfel de metodă poartă numele de “fat interface” sau “polluted interface” și poate aduce un comportament necorespunzător aplicației.

Acest principiu ne învață cum să scriem interfețe într-un mod cât mai eficient. Când proiectăm o aplicație și scriem propriile noastre interfețe trebuie să avem grijă să adăugăm doar metodele care ar trebui să se afle acolo, altfel clasa ce va implementa interfața va fi obligată să implementeze toate metodele, ceea ce oferă un design prost.

Există cel puțin 2 posibilități de evitare a încălcării acestui principiu:

- “Separarea prin delegare”: În acest caz se utilizează în design-ul aplicației Adapter Pattern care va delega doar către interfața necesară. Problema este rezolvată, dar într-un mod mai puțin elegant, deoarece se va crea

de fiecare data un nou obiect rezultand timp si memorie in plus. In cazul aplicatiilor cu sisteme de control in timp real este foarte important acest aspect si trebuie tratat corespunzator. • O alternativa reprezinta “Separarea prin mostenire multipla” . Aceasta abordare este mai eleganta si rezolva problema consumului de memorie si de timp, dar uneori este o metoda mai laborioasa pentru ca lantul de mostenire ar putea fi unul mai complex. Ambele posibilitati au avantaje si dezavantaje, iar in functie de aplicatie ne putem orienta catre una din alternative.

4 Laboratory work implementation

4.1 Analiza lucrării de laborator

<https://github.com/MarinaJechiuTI154/APPO/>

Pentru realizarea sarcinilor înaintate spre îndeplinire a fost utilizat limbajul Java, un limbaj specific paragmei programării orientate pe obiecte. Pentru o structurare logică mai bună, fiecare funcțional a fost integrat într-un pachet separat.

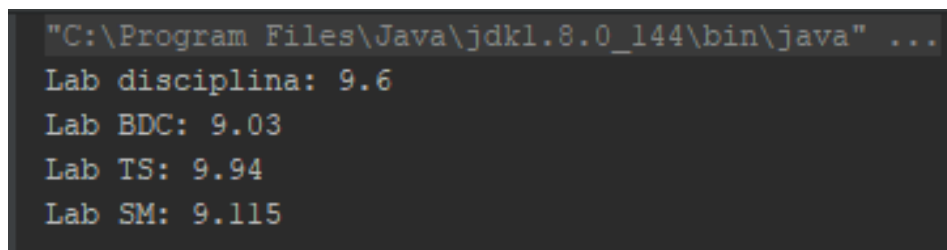
Tema aleasă pentru crearea claselor este o aplicație o aplicție de calculare a mediilor diferitor disciplini. Aceasta calculează media semestrială, conform diferitor criterii stabilite de profesor. Sunt luate în calcul notele testelor, laboratoarele și numărul absențelor.

Pentru respectarea principiului Liskov Substitution a fost creată o clasă care calculează media conform proporției 30 - 30 - 40. Aceasta calculează media unei discipline doar știind valorile atestărilor și examenului, sau media unei discipline concrete descrise de regulile interne ale acesteia.

Pentru a exemplifica funcționalitatea acestor obiecte, cât și relațiile dintre acestea au fost create obiecte concrete. Astfel, au fost calculată media mai multor discipline.

În figurile de mai jos sunt prezentate imagini ce exemplifică funcționalitatea aplicației.

4.2 Imagini



```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...  
Lab disciplina: 9.6  
Lab BDC: 9.03  
Lab TS: 9.94  
Lab SM: 9.115
```

Figure 4.1 – Afișarea comenzilor în proces de realizare

Concluzie

Sarcina de bază de implementare a celor 2 principii Dependency Inversion și Liskov Substitution a fost respectată și implementată. Acestea au fost implementate într-o aplicație de calcul al mediilor semestriale. Funcționalitatea softului a fost arătată în cadrul clasei `CalculareMedieTest`.