Cerințe proiect laborator POO

Cerințe etapa 1 | Cerințe etapa 2

Prezentare generală

Nota voastră la laboratorul de POO se acordă pe baza unui **proiect individual**, dezvoltat pe parcursul primelor 13 săptămâni din semestru[1]. Proiectul va fi evaluat în **trei etape**, cu cerințe și termene distincte pentru fiecare[2].

Proiectul constă în dezvoltarea unei **aplicații C++** în care să folosiți cât mai mult paradigma de **programare orientată pe obiecte**. Va trebui să definiți atât clase care să reprezinte entitățile gestionate de aplicația voastră, cât și metodele/funcțiile corespunzătoare care să implementeze „logica de business”.

Notare

Pentru fiecare etapă, veți primi **o notă de la 1 la 12**. Nota 10 se acordă pentru implementarea corectă a tuturor **cerințelor de bază**, putând primi până la două puncte în plus pentru **implementări deosebite** sau pentru rezolvarea **cerințelor suplimentare**.

Nota finală este **media** notelor de pe fiecare etapă. Trebuie să aveți minim **nota 5** ca să promovați laboratorul și să puteți participa la colocviu/examen.

Alegerea temei

Recomandarea mea este să alegeți o temă care vă pasionează, care să vă motiveze să lucrați la proiect pe tot parcursul semestrului.

Exemple de teme

- Gestiunea școlarității (studenți, note, discipline, profesori etc.)
- Magazin online (produse, cumpărători, comenzi, reduceri etc.)

[1] În ultima săptămână veți da colocviul (test pe calculator), până la acel moment trebuie să aveți situația la laborator încheiată.

[2] Cerințele permit continuarea programului din etapa anterioară, adăugând funcționalități în plus la codul existent.

- Gestiunea unei clinici/unui spital (medici, pacienți, consultații, medicamente etc.)
- Gestiunea resurselor umane (firmă, angajați, echipe, salarii etc.)
- Joc video cu interfață text sau grafică (jucător, item, monstru, nivel/hartă etc.)
- Aplicație de project management (proiect, task, echipă, membru al echipei etc.)
- Bibliotecă (carte, autor, categorie, împrumut etc.)
- Music player (album, artist, melodie, playlist etc.)
- Grădină zoologică (animale, bilete, hrană pentru animale etc.)
- Formula 1 (echipe, mașini, piloți, raliuri etc.)
- Cinematograf (filme, actori, vizionări, bilete etc.)
- Aplicație de banking (clienți, conturi, tranzacții etc.)

Puteți alege orice alt subiect doriți. Indiferent de tema aleasă, va **trebui să o confirmați** cu mine (în persoană când veniți la laborator, sau pe e-mail/Teams dacă nu puteți ajunge).

De preferat ar fi să vă alegeți teme distincte. Cel mult doi studenți pot avea o temă identică/similară. În acest caz, trebuie să aveți o structură diferită a claselor.

Recomandare: după ce v-ați ales tema și ați confirmat-o, puteți începe prin a vă face o „schiță" cu clasele de care veți avea nevoie, ce date și metode vor reține fiecare și care vor fi legăturile dintre ele. Puteți face asta pe hârtie sau folosind o aplicație cum ar fi Excalidraw.

Recomandare: familiarizați-vă cât mai curând cu și începeți să folosiți debuggerul din mediul vostru de lucru. O să vă petreceți 20% din timp scriind efectiv cod și peste 80% din timp încercând să înțelegeți de ce nu funcționează cum v-ați fi așteptat.

Cerințe etapa 1

Criterii generale

- Toate clasele și funcționalitățile implementate trebuie să fie **apelate/testate** (direct sau indirect) din `main` (să nu aveți cod nefolosit/inaccesibil în proiectul vostru). Clasele/metodele care nu sunt utilizate sau la care nu se face referire nicăieri **nu vor fi luate în considerare**. Încercați să definiți/implementați doar elementele de care aveți nevoie (la colocviu nu veți avea timp să implementați toate metodele posibile).
- Evitați folosirea **variabilelor globale**. Folosiți variabile locale sau date membru statice în clase.
- Păstrați codul curat:
  - Folosiți nume de variabile/funcții/tipuri de date cu sens pentru oameni (e.g. `nr_angajati` în loc de `n`).
  - Folosiți formatarea automată oferită de editorul vostru de text și încercați să păstrați un stil uniform.
  - Preferați folosirea mai multor clase/metode de dimensiuni reduse, decât o singură clasă/metodă foarte lungă.

Versionarea codului

- Codul sursă **trebuie** să fie încărcat pe GitHub. Puteți să creați un nou repository gol sau să folosiți acest template.
- În cazul în care repository-ul vostru nu este accesibil public, va trebui să îmi dați și mie **acces de citire**. Instrucțiunile pentru cum puteți adăuga colaboratori la un repo se găsesc aici. Mă puteți adăuga prin username (`GabrielMajeri`) sau prin e-mail (`constantin.majeri@s.unibuc.ro`).
- Repository-ul vostru trebuie să aibă un fișier `.gitignore`, pentru a nu include accidental fișierele compilate / binare în Git. Puteți găsi aici un exemplu de fișier `.gitignore` pentru C++.
- **Recomandare:** păstrați commit-urile concise și independente. Găsiți aici un set de bune practici pentru commit-urile de Git.

Documentație

- În repository-ul de pe GitHub trebuie să aveți și un fișier `README`, de preferat formatat cu Markdown, în care să includeți cel puțin:
  - **Numele** proiectului
  - **Tema** aleasă și o scurtă descriere a aplicației voastre
  - O listă cu **funcționalitățile** pe care le are aplicația voastră la momentul respectiv (ex.: „poate să citească și să rețină o listă de angajați", „poate calcula prețul mediu al produselor aflate în stoc" etc.)
- **Recomandare:** actualizați acest fișier pe parcurs ce dezvoltați proiectul. Vă va fi mult mai ușor să-l prezentați altor persoane.

Clase

- Trebuie să respectați **principiul encapsulări** (nu aveți voie cu date membre publice). Metodele pot fi publice sau private, în funcție de rolul lor.
- Trebuie să definiți **minim 3-4 clase**, relevante pentru tema aleasă. Acestea trebuie să fie corelate prin **compunere** (ex. să aveți o dată membru de tip `Adresă` în clasa `Contact`, să aveți un vector de `Angajat` în clasa `Companie` etc.)

Constructori

- Toate clasele trebuie să aibă definite minim un **constructor de inițializare** (cu sau fără parametri, în funcție de specificul clasei).
- **Minim o clasă** trebuie să aibă definit complet și corect **constructorul de copiere**, `operator=` și **destructorul** (în care puteți de exemplu să resetați valorile câmpurilor din acea clasă).

Metode

- **Inițializați cel puțin o instanță** din fiecare dintre clasele definite, folosind o metodă/funcție de citire sau suprascriind `operator>>`. De preferat este să citiți datele din fișier (este mai rapid și pentru voi). Dacă alegeți să le citiți de la tastatură, salvați datele de intrare într-un fișier text și luați-le cu copy & paste de acolo când aveți nevoie.
- Pentru cel puțin una dintre clase trebuie să aveți **supraîncărcat** `operator>>` pentru citire.

- Toate clasele trebuie să aibă **supraîncărcat** `operator<<` pentru afișare.
- Definiți **minim 2 getteri și 2 setteri** pentru datele membru (pot fi pentru câmpuri diferite, din clase diferite).
- **Minim 2 metode de „logică de business”** care să aibă sens pentru tema voastră (e.g.: calculează prețul unui produs aplicând o reducere, returnează salariul mediu al angajaților din firmă etc.)

Bonus

- Folosiți modificatorul `const` în toate situațiile în care are sens (e.g. getteri, funcții de afișare, parametrii care nu se modifică etc.)
- Implementați „teste” pentru codul vostru, ca să vă asigurați că funcționează cum trebuie. Puteți utiliza funcția utilitară `assert`.

Cerințe etapa 2

Pentru a implementa aceste cerințe, puteți continua/extinde codul de la etapa 1, actualizându-l pentru a folosi conceptele noi.

Proiectul vostru trebuie să respecte în continuare **Criteriile generale** de la prima parte (cod curat, care respectă encapsularea, fără variabile globale, fără clase/-metode care nu sunt folosite/apelate nicăieri), precum și indicațiile referitoare la **Versionarea codului** (continuați să încărcați ce lucrați pe Git) și la **Documentație** (actualizați corespunzător `README`-ul).

Moștenire

- Definiți **minim două ierarhii diferite de moștenire** în cadrul programului vostru (două ierarhii de moștenire sunt considerate diferite dacă nu au aceeași clasă de bază în comun).
- **Minim o dată membru** și **minim o metodă** care să aibă modificatorul de acces `protected` (în mod util, să fie accesate/apelate dintr-o clasă care le moștenește).
- **Cel puțin o situație** în care să apelați constructorul (cu parametri) al clasei de bază, folosind lista de inițializare din constructorul clasei derivate.

Metode virtuale și clase abstracte

- Definiți și extindeți (moșteniți) **minim o clasă abstractă** (poate avea date membru, dar are cel puțin o metodă pur virtuală).
- Definiți **cel puțin 2 metode virtuale** care să fie suprascrise în clasele moștenitoare. Pot fi pur virtuale sau cu o implementare implicită. Se iau în considerare și metodele definite la celelalte subpuncte, exceptând destructorii virtuali.

Polimorfism la execuție

- Identificați și marcați prin câte un comentariu **minim 2 instanțe** în care să aibă loc polimorfism la execuție (*dynamic dispatch*) în proiectul vostru (e.g. apelul unor metode virtuale prin intermediul unor pointeri/referințe către clasa de bază).
- Identificați și marcați prin câte un comentariu **minim 2 instanțe** de *upcasting* în codul vostru (e.g. atribuirea unor obiecte de tipul unor clase moștenite

la pointeri/referințe către clasa de bază).

Excepții

- Definiți **minim un tip de excepție custom**, care să extindă clasa `exception` din biblioteca standard.
- Aruncați excepții în **minim 2 funcții/metode diferite** (folosiți tipul de excepție definit de voi sau pe cele din biblioteca standard).
- Implementați **minim un bloc** `try...catch` care să prindă o excepție definită și generată de voi (cu specificarea explicită a tipului excepției capturate) și să o trateze într-un fel (să afișeze un mesaj, să reîncerce operațiunea, să arunce o altă excepție etc).

Variabile și metode statice

- Definiți și inițializați o variabilă membru statică în **cel puțin o clasă**.
- Implementați **cel puțin două metode statice** în clasele voastre (din care **cel puțin una** trebuie să acceseze/folosească variabila statică definită la subpunctul anterior).

Bonus

- Separați **declarațiile** și **implementările** din programul vostru folosind fișiere header (`.h`/`.hpp`) și sursă (`.cpp`) distincte. Ar trebui să aveți câte un fișier header și un fișier sursă pentru fiecare clasă din programul vostru.
- Identificați și implementați o situație de **moștenire în diamant** în proiectul vostru (trebuie să aibă sens relativ la tema aleasă). Moștenirea în diamant se referă la a avea o clasă de bază, pe care o moștenesc cu `virtual` două clase distincte, iar apoi aveți o clasă care moștenește ambele clase intermediare.