Laboratorul 8

CLI Menus

# Takeaways

* Vom învăța cum să creăm un meniu interactiv pentru o aplicație de consolă

# Exerciții

Cel mai simplu meniu pe care îl puteți implementa funcționează în acest mod:

<https://youtu.be/cDccI44W5Lo>

Există și variante mai bune, deci la finalul zilei puteți fii creativ și să vă creați cod mai bun. Excellence is a habit! :)

## Exercițiul 1

Implementați propriul program care se va comporta ca în [clipul de mai sus](https://youtu.be/cDccI44W5Lo). Pentru a ne aminti de interfețe și override, recomand să începeți cu acest cod:

*// Vom porni de la intrebarile:*

*// Ce ar trebui sa faca un meniu interactiv? Ce pasi sunt necesari pentru a-l folosi.*

*// Orice meniu va putea:*

*// \* lista optiunile*

*// \* alege corect una dintre optiuni*

*// \* va putea fi rulat la infinit (=interactiv din "meniulinteractiv")*

*class* BaseMenu {

*public*:

*virtual void* listOptions() {}

*virtual int* chooseOption(*int* first, *int* last) {}

*virtual void* mainLoop() {}

};

Iar scopul vostru este să creați o clasa derivată/ moștenitoare care rezolvă exercițiul.

Funcția main() pe care o veți rula este:

*int* main() {

SimpleMenu menu;

menu.mainLoop(); *// păstrăm codul simplu: funcția mainLoop „ruleaza” meniul la infinit.*

}

## Exercițiul 2 - Meniu interactiv pentru operații CRUD

*(Create/ Read/ Update/ Delete)*

***Aveți grijă să citiți și cerința exercițiului 3, pentru a găsi o soluție astfel încât codul de aici să poată fi folosit și la 3, prin compunere de clase.***

Creați un meniu care pe lângă faptul că moștenește BaseMenu, va avea acces și la un vector<Building>. Building este o clasă simplă, care moștenește IoBase ca să poată fi citită/ afișată ușor, și conține date despre culoare și numele proprietarului.

Implementați următoarele opțiuni:

1. Adaugă o nouă clădire.
2. Afișează toate clădirile
3. Șterge una dintre clădirile reținute.
4. Modifică una dintre clădirile existente

Bonus +10p: utilizați vector<shared\_ptr<Building> >, precum în **laboratorul 6**

Pentru bonus se cere:

1. Crearea claselor House și Apartment. House conține și câmpul **cntFloors**, iar Apartment **floor.**
2. Inițializarea cu date în main (adică de forma int[] v = {2,3,4};) și de;
3. O funcție globală care citește un obiect de tipul **House** sau **Apartment** și returnează o valoarea de tipul shared\_ptr<Building>

## Exercițiul 3 - Bază de date cu mai multe entități

Aici aveam o singura intrebare. Cum putem refolosi codul de mai sus pentru a extinde pentru mai multe tipuri de date pe care face CRUD. E.g. Building si Owner in acelasi meniu.

*De aici în jos avem exerciții mai “competitive”:*

# Exercițiul 4 - Meniuri și submeniuri (nested)?

Revenim la SimpleMenu, dar vom adăuga în toate meniurile(deci in BaseMenu) un cămp de tipul string, denumit **label**.

Vom crea un tip de meniu care seamănă foarte bine cu meniurile reale, pentru a cunoaște puțin implementarea.

Știți foarte clar ca dacă dați în CLion/ alt editor pe **File -> New** se deschide încă un șir de meniuri, cu opțiuni precum Project, C++ Class etc

Păi asta înseamnă că avem un tip de meniu compus (ComposedMenu) care conține o listă cu mai multe meniuri. Fiecare dintre acestea pot fi ori simple, ori compuse.

### 

### Mai multe soluții

Soluția 1 - In care in meniurile compuse vom folosi

**map<string, shared\_ptr<BaseMenu> >**

Mai exact, string-ul va fi eticheta meniului, iar shared\_ptr<BaseMenu> va fi submeniul cu acea eticheta.

Solutia 2 - daca nu folosim map, putem folosi in schimb

**vector<pair<string, shared\_ptr<BaseMenu> >**

Sau să ținem cont că avem deja câmpul:

**string label;// cea mai simplă variantă, ca tipuri de date, dar nu învățăm map :(**

Deci nu mai are rost să folosim map, care este util pentru simplitatea în a afișa toate opțiunile, în a alege în funcție de denumire o opțiune, fără să le alegem prin indice.

# Mai clar

***Cum folosim meniuri compuse, chiar dacă fiecare are propriul mainLoop() ?***

Un meniu compus este de fapt un meniu în care sunt afișate submeniurile, opțiunea de încheiere a programului, și opțional alte alegeri/ opțiuni posibile (precum “detaliază opțiuni” etc).

Să spunem ca avem meniul compus cu label-ul File, si doua submeniuri simple: Create și Open.

Adica o compunere de tipul:

* File
  + Create
    - C++ Class
    - Text File
  + Open
    - Browse
  + Delete File..

Unde cu cerc negru avem meniul compus, cu cerc alb avem cele doua meniuri simple, iar cu patrat negru avem opțiunile propriu size ale meniului:

**C++ Class**

**Text File**

**Browse**

**Delete File..**

Asadar, prin navigarea meniului vom putea efectua oricare din cele 4 operatii. Pentru simplitate (deoarece acum nu ne intereseaza sa codam un FileSystem), tot ce vom face in functii va fi sa afisam mesaje precum “Am creat cu succes o clasa C++”, “Am deschis unul dintre fisierele existente” etc.

# Exercițiul 5 - Ați mai folosit **callbacks**?

*Acesta este doar un experiment, deoarece deși este interesant să creăm mai flexibil meniul, probabil nu își merită bătaia de cap. Sigur nu în timpul colocviului :).*

Pe scurt, vom automatizam chiar mai mult crearea unui CrudMenu.

Cerinta:

Se cere să implementați un meniu care poate fi creat direct în acest mod:

#include<functional>

#include<vector>

#include<string>

#include<map>

*using namespace* std;

*class* Building : *public* IoBase {

*private*:

string color, owner;

*int* height;

*// TODO constructori si getteri:*

};

*class* CrudMenu : *public* BaseMenu {

*private*:

string label;

map<string, function<*void*(*const* vector<Building> &)> > options;

*public*:

};

*// PUTEM folosi functii globale care sa acceseze sirul de elemente*

*void* sorteaza(vector<Building>)

*int* main() {

*// PUTEM folosi lambda functions pentru a crea metodele*

*// function<TIP\_RETURNAT(LISTA\_TIP\_ARGUMENTE)>*

function<*void*(*const* vector<Building> &)> afiseaza = [](*const* vector<Building> &v) -> *void* {

*for* (*auto* building: v) {

cout << building << '\n';

}

};

*// ASA cream un map*

*// map<int,int> m = {*

*// {2,3},*

*// {3,4}*

*// };*

CrudMenu menu(

{ *// vectorul de cladiri (culoare, proprietar, inaltime)*

Building("red", "Gabriel", 7),

Building("blue", "Penelope", 4),

},

{ *// map-ul care contine etichetele optiunilor si functiile rulata*

{"Afiseaza", afiseaza},

{"Sorteaza", sorteaza}

}

);

menu.mainLoop();

}

Bonus +20p: putem automatiza un meniu de tip CRUD, astfel încât să funcționeze pentru orice tip de clase.

# 