Laborator Final

Tema: Recapitulare Generală. Smart Home System 🏠

Poveste: The Architect of Tomorrow

Felicitări! Ai ajuns la finalul stagiului de pregătire. Compania **FutureTech** te-a angajat pentru a dezvolta nucleul software al noului lor produs revoluționar: **Casa Inteligentă (Smart Home Hub)**.

Sistemul actual este haotic: fiecare dispozitiv (bec, termostat, cameră video) este gestionat separat, ducând la cod duplicat și greu de întreținut. Misiunea ta este să creezi o arhitectură orientată pe obiecte, robustă și scalabilă, care să poată gestiona orice tip de dispozitiv într-un mod unitar.

Va trebui să demonstrezi că stăpânești conceptele de **abstractizare**, **moștenire**, **polimorfism**, **supraîncărcarea operatorilor** și gestionarea colecțiilor folosind **STL**.

Scopul laboratorului

- Organizarea modulară a codului (fișiere Header și Source).
- Consolidarea cunoștințelor despre **Clase Abstracte** și **Metode Virtuale Pure**.
- Utilizarea **Polimorfismului** pentru procesarea uniformă a obiectelor.
- Gestionarea resurselor folosind containere **STL** (vector).
- Utilizarea membrilor **Statici** și **Supraîncărcarea operatorilor**.

Cerințe de implementare (9p + 1p oficiu)

IMPORTANT: Organizarea Codului

- Pentru fiecare clasă implementată, trebuie să creați o pereche de fișiere: `NumeClasa.h` (declarația clasei) și `NumeClasa.cpp` (implementarea metodelor).
- Funcția `main` se va afla într-un fișier separat, `main.cpp`.

1. **Arhitectura de Bază: Dispozitivele (4p)**
   - Definiți o clasă abstractă `Device` (în `Device.h/cpp`) care să conțină:
     - Un membru `string name` și unul `int id`.
     - Un constructor care inițializează aceste câmpuri.
     - O metodă pur virtuală `void displayStatus() const`.
     - Un **destructor virtual** (esențial pentru ștergerea corectă prin pointeri la bază).
   - Implementați două clase derivate (în fișiere proprii):
     - `SmartLight`: adaugă un câmp `float brightness` (0-100). Suprascrieți `displayStatus` pentru a afișa numele și intensitatea luminii.
     - `Thermostat`: adaugă un câmp `float temperature`. Suprascrieți `displayStatus` pentru a afișa numele și temperatura curentă.
2. **Hub-ul Central: Gestiunea Colecțiilor (3p)**
   - Creați clasa `SmartHome` (în `SmartHome.h/cpp`) care funcționează ca un manager.
   - Aceasta trebuie să conțină o listă de dispozitive. Folosiți **STL**:

```
#include <vector>
#include "Device.h"
// ...
std::vector<Device*> devices;
```

   - Implementați metoda `void addDevice(Device* dev)` care adaugă un dispozitiv în sistem.
   - Implementați destructorul clasei `SmartHome` care să șteargă memoria pentru toate dispozitivele stocate în vector, pentru a evita memory leaks.
3. **Operatori și Membri Statici (2p)**
   - **Contor Global:** Adăugați în clasa `Device` un membru static `static int deviceCount` care să țină evidența numărului total de dispozitive active.
     - Incrementați contorul în constructorul clasei de bază.
     - Decrementați contorul în destructorul clasei de bază.
     - Afișați numărul total de dispozitive la finalul programului `main`.
   - **Supraîncărcare operator «:** Supraîncărcați operatorul de inserție în flux (`«`) pentru clasa `SmartHome`.
     - Acesta trebuie să afișeze titlul "Raport Smart Home:" urmat de statusul tuturor dispozitivelor din casă (iterând prin vector și apelând `displayStatus`).
     - În `main`, afișarea întregului sistem trebuie să se facă simplu: `cout « myHome « endl;`

Ghid Conceptual Recapitulativ

De ce Destructor Virtual?

Atunci când ștergem un obiect derivat printr-un pointer la clasa de bază (`Device* d = new SmartLight(...); delete d;`), dacă destructorul din bază nu este virtual, se va apela doar destructorul bazei. Destructorul derivatei nu se execută, ceea ce poate duce la scurgeri de memorie.

Polimorfismul în STL

Containerele STL (precum `vector`) stochează elemente de același tip. Pentru a stoca o ierarhie de clase (`SmartLight` și `Thermostat` la un loc), trebuie să stocăm **pointeri la clasa de bază** (`vector<Device*>`). Astfel, la apelul metodelor virtuale, se va decide la runtime care implementare să fie executată (Late Binding).

Compilare și verificare

Deoarece aveți mai multe fișiere, trebuie să le compilați pe toate împreună pentru a crea executabilul.

```
# Compilare (listând toate fișierele sursă)
g++ -o final_lab main.cpp Device.cpp SmartLight.cpp Thermostat.cpp SmartHome.cpp

# SAU (folosind wildcard - compilează toate fișierele .cpp din folder)
g++ -o final_lab *.cpp

# Rulare
./final_lab
```

Finalizare și Încărcare GitHub

Acesta este ultimul laborator. Asigurați-vă că ați parcurs toate cerințele pentru a valida cunoștințele acumulate. Studenții pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului.

Pași pentru încărcarea soluției finale:

(a) Accesați linkul **GitHub Classroom** pentru Laboratorul Final:

`https://classroom.github.com/a/pAu-bWfK`

(b) Clonați repository-ul creat automat.

(c) Încărcați sursele pe branch-ul main:

```
git add .
git commit -m "Lab 13 - Smart Home Implementation"
git push origin main
```