Laborator 9

Tema: Clase Abstracte și Interfețe

Poveste

O platformă modernă de streaming dorește să își actualizeze arhitectura software pentru a suporta diverse tipuri de conținut multimedia. Platforma gestionează elemente generice de tip **Media**, însă fiecare tip specific (ex. **Film**, **Melodie**, **Podcast**) se comportă diferit atunci când este redat. De exemplu, un film necesită încărcarea subtitrărilor, în timp ce o melodie afișează versurile.

De asemenea, platforma dorește să integreze funcționalități de social media. Doar anumite elemente pot fi distribuite (*shared*) pe rețelele sociale, indiferent dacă sunt fișiere media sau profiluri de utilizator.

Pentru a modela acest sistem, vom folosi **clase abstracte** pentru a defini comportamentul de bază al unui element multimedia (care nu poate fi instanțiat direct) și **interfețe** pentru a oferi capabilitatea de distribuire (`IShareable`) unor clase disparate.

Scopul laboratorului

- înțelegerea conceptului de **funcție virtuală pură**;
- definirea și utilizarea **claselor abstracte**;
- implementarea conceptului de **interfață** în C++;
- mecanismul de funcționare al **polimorfismului** prin pointeri la clase de bază.

Structura fișierelor

- **IShareable.h** — Interfață pură ce definește contractul pentru distribuire.
- **Media.h** / **Media.cpp** — Clasă abstractă ce conține atribute comune (titlu) și metode virtuale pure.
- **Film.h** / **Film.cpp** — Clasă concretă ce moștenește `Media` și implementează `IShareable`.
- **main.cpp** — Testarea polimorfismului.

Fișierul `IShareable.h`

```cpp
#pragma once
#include <iostream>

// Aceasta este o INTERFATA.
// Contine doar metode virtuale pure si un destructor virtual.
class IShareable {
public:
    virtual void share() = 0; // Metoda virtuala pura
    virtual ~IShareable() {}
};
```

Fișierul `Media.h`

```cpp
#pragma once
#include <iostream>

// Aceasta este o CLASA ABSTRACTA.
class Media {
protected:
    char* titlu; // Resursa alocata dinamic
    int durata;  // Durata in secunde

public:
    // Constructor si Destructor (virtual!)
    Media(const char* t, int d);
    virtual ~Media();

    // Metode virtuale pure -> Obliga derivatele sa le implementeze
    virtual void reda() = 0;
    virtual void afiseazaDetalii() = 0;

    // Metode concrete (care au implementare aici)
    int getDurata() const;
};
```

Fișierul `Film.h`

```h
#pragma once
#include "Media.h"
#include "IShareable.h"

// Mostenire Multipla: Film este un Media si este IShareable
class Film : public Media, public IShareable {
private:
    int anAparitie;

public:
    // Constructor
    Film(const char* titlu, int durata, int an);

    // Destructor
    ~Film();

    // Implementarea obligatorie a metodelor din Media
    void reda() override;
    void afiseazaDetalii() override;

    // Implementarea obligatorie a metodei din IShareable
    void share() override;
};
```

Explicații suplimentare

- **Funcții Virtuale Pure** — Sunt funcții declarate cu `= 0` (ex: `virtual void f() = 0;`). Acestea nu au implementare în clasa de bază și **trebuie** suprascrise în clasele derivate pentru ca acelea să poată fi instanțiate.
- **Clase Abstracte** — O clasă care conține cel puțin o funcție virtuală pură devine abstractă. Nu se pot crea obiecte de tipul clasei abstracte (ex: `Media m;` este ilegal), dar se pot declara pointeri (`Media* m;`).
- **Interfețe în C++** — C++ nu are cuvântul cheie `interface` (ca Java). O interfață se simulează printr-o clasă abstractă care conține **doar** funcții virtuale pure și nu are atribute (membri de date).
- **Destructorul Virtual** — Este critic într-o ierarhie de clase. Dacă ștergem un obiect derivat printr-un pointer la clasa de bază (`Media* m = new Film(...); delete m;`), destructorul bazei trebuie să fie virtual pentru a se apela și destructorul derivatei.

Cerințe (9p + 1p oficiu)

1. **Definire Interfață și Clasă Abstractă (3p)**
   - (1p) Implementarea interfeței `IShareable` (doar header).
   - (2p) Implementarea clasei abstracte `Media`. Atenție la **Regula celor Trei** (destructor, cc, op=) deoarece avem membrul `char* titlu`. Destructorul trebuie să fie `virtual`.
2. **Implementare Clasă Concretă (3p)**
   - (1p) Definirea clasei `Film` care moștenește **atât** `Media` cât și `IShareable` (moștenire multiplă).
   - (2p) Implementarea metodelor virtuale pure: `reda()` (afișează "Redare film: [titlu]"), `afiseazaDetalii()` și metoda `share()` din interfață.
3. **Testare și Polimorfism (3p)**
   - (1p) În `main`, creați un vector de pointeri la `Media` (ex. `Media* playlist[3]`) și populați-l cu obiecte de tip `Film`.
   - (1p) Iterați prin vector și apelați metodele `reda()` și `afiseazaDetalii()`. Observați apelul polimorfic al metodelor din clasa abstractă.
   - (1p) Demonstrați funcționarea interfeței: Declarați un pointer de tip `IShareable*` căruia îi atribuiți adresa unui obiect `Film` și apelați metoda `share()`. Acest lucru demonstrează că obiectul poate fi tratat strict prin prisma interfeței sale.
4. **(1p) Punct din oficiu.**

Compilare și verificare

```
g++ -g main.cpp Media.cpp Film.cpp -o lab9
./lab9
valgrind ./lab9
```

Bonus GitHub Classroom

Studenții pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului, respectiv **1p bonus** dacă încarcă mai mult de jumătate dintre ele.

Pași pentru încărcarea laboratorului:

(a) Accesați cursul de pe OCW – Programare Orientată pe Obiecte, secțiunea **Resurse** → **Bonus GitHub laborator**.

(b) Acolo veți găsi linkul **GitHub Classroom** pentru Laboratorul 9:

`https://classroom.github.com/a/1HmGkW56`

(c) După acceptare, veți primi propriul repository, ex. `laborator-9-NumePrenume`.

(d) Clonați repository-ul local:

```
git clone git@github.com:Laborator-POO-2025-2026/laborator-9-NumePrenume.git
cd laborator-9-NumePrenume
```

(e) După ce ați adăugat fișierele corespunzătoare (`Media.h`, `Film.h`, etc.), urmează comenzile:

```
git add .
git commit -m "Implementare laborator 9 - Clase Abstracte"
git push origin main
```