Laborator 10

Tema: Vectori de obiecte neomogene

Poveste

Biblioteca Centrală dorește să modernizeze sistemul de inventariere a cărților. Fondul de carte este divers, conținând atât literatură beletristică (**Romane**), cât și materiale didactice (**Manuale**). Deși toate sunt cărți și au un titlu și un preț de inventar, detaliile specifice diferă: un roman este clasificat după un gen literar (ex: "SF", "Dramă"), în timp ce un manual este asociat unei materii specifice (ex: "Matematică", "Fizică"). Administratorul bibliotecii dorește o aplicație care să stocheze toate publicațiile într-un singur catalog digital și să poată afișa detaliile complete ale fiecărei cărți, indiferent de tipul acesteia. Vom utiliza un **vector de obiecte neomogene** (pointeri la o clasă de bază abstractă `Carte`) pentru a gestiona catalogul uniform, bazându-ne pe polimorfism pentru afișare.

Scopul laboratorului

- recunoașterea și utilizarea unui **vector de obiecte neomogene**;
- alocarea și dezalocarea memoriei pentru un vector de pointeri (`Type**`);
- înțelegerea necesității unui **destructor virtual pur** într-o ierarhie de clase;
- supraîncărcarea **operatorului de afișare** (`«`) folosind metode virtuale.

Structura fișierelor

- **Carte.h** / **Carte.cpp** — Clasă abstractă (Interfață) ce definește contractul pentru orice publicație.
- **Roman.h** / **Roman.cpp** — Clasă concretă pentru literatură (are gen literar).
- **Manual.h** / **Manual.cpp** — Clasă concretă pentru studiu (are materie).
- **main.cpp** — Gestionarea catalogului de cărți.

Fișierul `Carte.h`

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

class Carte {
protected:
    // Metoda virtuala pura pentru afisare
    virtual void afisare(std::ostream& out) const = 0;

public:
    // Destructor virtual pur (obligatoriu pentru curatarea corecta a derivatelor dar
    ↪  cere si o implementare concreta goala in fisierul Carte.cpp)
    virtual ~Carte() = 0;

    // Metode virtuale pure (Interfata)
    virtual float getPret() const = 0;
    virtual char* getTitlu() const = 0;

    // Supraincarcarea operatorului << (functie friend)
    // Se implementeaza in Carte.cpp
    friend std::ostream& operator<<(std::ostream& out, const Carte& c);
};
```

Fișierul `Roman.h`

```cpp
#pragma once
#include "Carte.h"

class Roman : public Carte {
private:
    float pret;
    char* titlu; // Alocare dinamica
    char* genLiterar; // Alocare dinamica (ex: "SF", "Politist")

protected:
    // Implementarea specifica a afisarii
    void afisare(std::ostream& out) const override;

public:
    Roman(float pret, const char* titlu, const char* gen);
    ~Roman(); // Dezaloca titlu si genLiterar

    float getPret() const override;
    char* getTitlu() const override;
};
```

Fișierul `Manual.h`

```h
#pragma once
#include "Carte.h"

class Manual : public Carte {
private:
    float pret;
    char* titlu; // Alocare dinamica
    char* materie; // Alocare dinamica (ex: "Programare")

protected:
    void afisare(std::ostream& out) const override;

public:
    Manual(float pret, const char* titlu, const char* materie);
    ~Manual(); // Dezaloca titlu si materie

    float getPret() const override;
    char* getTitlu() const override;
};
```

Ghid pentru `main.cpp`

```cpp
#include <iostream>
#include "Roman.h"
#include "Manual.h"

int main() {
    int nrCarti = 4;
    // 1. Alocare vector de pointeri la clasa de baza (Carte**)
    // Codul tau aici...
    // 2. Initializare vector cu obiecte diverse (Upcasting implicit)
    // biblioteca[0] = new Roman(... , ... , ...);
    // biblioteca[1] = new Manual(... , ... , ...);
    // ...
    // 3. Afisare polimorfica
    // Se va apela operatorul << care cheama metoda virtuala afisare()
    // for (...) { cout << *biblioteca[i] << endl; }
    // 4. Dezalocare memorie
    // Pasul A: Stergem obiectele concrete (se apeleaza destructorii virtuali)
    // for (...) { delete biblioteca[i]; }
    // Pasul B: Stergem vectorul de pointeri
    // delete[] biblioteca;
    return 0;
}
```

Explicații suplimentare

- **Vectorul Neomogen** — Nu știm câte romane și câte manuale vom avea în bibliotecă. Folosind un vector de pointeri la baza 'Carte' ('Carte** biblioteca'), putem stoca adresele unor obiecte diferite ('new Roman(...)', 'new Manual(...)') în același container.
- **Virtualizarea Operatorului < <** — Operatorii prieteni (friend) nu sunt membri ai clasei, deci nu pot fi 'virtual'. Soluția elegantă este ca operatorul '«' să apeleze o funcție membră virtuală protejată ('afisare'). Astfel, 'cout' va "ști" să afișeze corect tipul cărții la runtime.
- **Curățarea Memoriei** — Când ștergem un vector de pointeri neomogeni, trebuie să parcurgem vectorul și să apelăm 'delete' pe fiecare element. Dacă destructorul bazei (' Carte') nu ar fi virtual, s-ar apela doar destructorul de bază, iar memoria alocată în clasele derivate (pentru 'titlu', 'genLiterar', 'materie') ar rămâne ocupată (memory leak).

Cerințe (9p + 1p oficiu)

1. **Clasa Abstractă și Afișarea (3p)**
   - (1.5p) Definiți clasa `Carte` cu metodele virtuale pure necesare.
   - (1.5p) Implementați operatorul `«` în cpp care să delege afișarea către metoda 'afisare()'. Oferiți o implementare goală destructorului pur ' Carte'.
2. **Clasele Derivate (3p)**
   - (1.5p) Implementați clasa `Roman`: constructor (copiere adâncă pentru 'char*' titlu și gen), destructor și logica de afișare specifică ("Romanul [titlu], genul [gen]...").
   - (1.5p) Implementați clasa `Manual` similar, gestionând memoria pentru 'titlu' și 'materie'.
3. **Gestiunea prin Vector Neomogen (3p)**
   - (1p) În `main`, creați un vector de pointeri `Carte**`.
   - (1p) Adăugați instanțe de `Roman` și `Manual` în acest vector.
   - (1p) Parcurgeți vectorul pentru a afișa cărțile, apoi eliberați corect toată memoria (ștergere elemente + ștergere vector).
4. **(1p) Punct din oficiu.**

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 10:

`https://classroom.github.com/a/oCC3Uas3`

(c) După acceptare, veți primi propriul repository, ex. `laborator-10-NumePrenume`.

(d) Clonați repository-ul local:

```
git clone git@github.com:Laborator-POO-2025-2026/laborator-10-NumePrenume.git
cd laborator-10-NumePrenume
```

(e) După ce ați adăugat fișierele corespunzătoare (`Carte.h`, `main.cpp`, etc.), urmează comenzile:

```
git add .
git commit -m "Implementare laborator 10 - Gestiune Biblioteca"
git push origin main
```