Laborator 4 – Particularitățile clasei

Tema: Studentul Cinefil 🎞 – Partea a II-a

Autor: ing. Eduard Donea

Poveste

Andrei a ajuns la partea a doua a proiectului său despre filme. După ce a învățat cum să gestioneze o colecție de obiecte, el descoperă acum nevoia de a copia și reasigna obiecte fără a produce erori de memorie. În același timp, observă că fiecare film ar trebui să aibă un cod unic, care nu se repetă niciodată — chiar dacă obiectele se distrug.

Scopul laboratorului

- folosirea constructorului de copiere și a operatorului de atribuire;
- diferențierea dintre `const` și `static`;
- înțelegerea conceptului de **unicitate a obiectelor** prin atribute `const`;
- implementarea unui contor comun tuturor obiectelor (membru `static`);
- verificarea corectitudinii alocării și eliberării memoriei cu `valgrind`.

Structura fișierelor

- **film.h** – declararea clasei `Film`;
- **film.cpp** – implementarea metodelor clasei;
- **main.cpp** – testarea funcționalităților.

Fișierul `film.h`

```
#ifndef FILM_H
#define FILM_H

class Film {
    char* titlu;
    char* gen;
    int durata;                         // in minute
    float nota;                         // nota acordata de student (0{10)

    const int codUnic;                  // valoare unica per obiect
    static int numarFilme;              // contor comun tuturor obiectelor

public:
    // Constructori
    Film();
    Film(const char* titlu, const char* gen, const int durata, const
    float nota);
    Film(const Film& other);            // constructor de copiere
    Film& operator=(const Film& other); // operator de atribuire

    // Destructor
    ~Film();

    // Getteri const
    char* getTitlu() const;
    char* getGen() const;
    int getDurata() const;
    float getNota() const;

    const int getCodUnic() const;

    // Setteri
    void setTitlu(const char* titlu);
    void setGen(const char* gen);
    void setDurata(const int& durata);
    void setNota(const float& nota);

    // Alte metode
    void afisare() const;
    static int getNumarFilme(); // metoda statica
};

#endif FILM_H
```

Fișierul `main.cpp` (model)

```cpp
#include "film.h"

int main() {

    // TODO 1: Creati un obiect f1 folosind constructorul cu parametri
    // si il afisati.

    // TODO 2: Creati un obiect f2 folosind constructorul de copiere.
    // Ex: Film f2(f1);

    // TODO 3: Testati operatorul de atribuire (=)
    // Ex: f3 = f1;

    // TODO 4: Cititi numarul de filme n si alocati un vector dinamic.
    // Ex: Film* v_filme = new Film[n]

    // TODO 5: Cititi datele fiecarui film (pret, durata, titlu, gen)
    // si completati vectorul folosind metoda set sau constructorul
    // cu parametri.

    // TODO 6: Afisati toate filmele citite si codurile lor unice.

    // TODO 7: Afișati numarul total de filme create folosind metoda statica.

    // TODO 8: Eliberati memoria si verificati cu Valgrind.
    // g++ -g main.cpp film.cpp -o lab4
    // valgrind ./lab4

    return 0;
}
```

Explicații suplimentare

- **const** — marchează variabile care nu se pot modifica după inițializare.

```
const int codUnic; // se setează doar în lista de inițializare
```

- **static** — aparține clasei, nu obiectului; are o singură copie comună.

```
static int numarFilme; // comun tuturor obiectelor
```

- Membrii **static** se inițializează în fișierul `.cpp`:

```
int Film::numarFilme = 0;
```

- Constructorul de copiere trebuie să aloce dinamic memorie nouă și să copieze conținutul:

```
// Exemplu:
Film::Film(const Film& f) : codUnic(++numarFilme) {

    this->titlu = new char[strlen(f.titlu) + 1];
    strcpy(this->titlu, f.titlu);

    this->gen = new char[strlen(f.gen) + 1];
    strcpy(this->gen, f.gen);

    this->durata = f.durata;
    this->nota = f.nota;
}

// Obs: Utilizarea lui "this" nu este necesara dar va ajuta sa
// vizualizati mai bine obiectele si cum se face atribuirea.
```

Cerințe (10p)

1. (3p) Implementați clasa `Film` cu atributele: `pret`, `durata`, `titlu`, `gen`.
2. (2p) Implementați constructorul de copiere și operatorul de atribuire.
3. (2p) Adăugați un atribut `const int codUnic` și un membru static `numarFilme`.
4. (1p) Afișați codul unic al fiecărui film.
5. (1p) Afișați numărul total de filme create.
6. (1p) Testați totul cu `valgrind`.

Compilare și verificare

```
g++ -g main.cpp film.cpp -o lab4
./lab4
valgrind ./lab4
```

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:

1. Accesați cursul de pe OCW – Programare Orientată pe Obiecte, secțiunea **Resurse** → **Bonus GitHub laborator**.
2. Acolo veți găsi linkul **GitHub Classroom** pentru Laboratorul 3:

   `https://classroom.github.com/a/e5IrDPgO`

3. După acceptare, veți primi propriul repository, ex. `laborator-3-NumePrenume`.
4. Clonați repository-ul local:

   ```
   git clone
   ↪  git@github.com:Laborator-POO-2025-2026/laborator-4-NumePrenume.git
   cd laborator-4-NumePrenume
   ```

5. După ce ați adăugat fișierele `film.h`, `film.cpp` și `main.cpp`, urmează comenzile:

   ```
   git add .
   git commit -m "Mesaj de commit"
   git push origin main
   ```

Atenție: laboratoarele copiate între studenți sunt considerate nevalide, iar niciunul dintre cei doi participanți nu va primi bonusul. Repository-urile sunt monitorizate automat de sistemul GitHub Classroom.