Laborator 4 – Particularitățile clasei

Tema: Studentul Cititor – Partea a II-a

Autor: ing. Eduard Donea

Poveste

Maria a ajuns la partea a doua a proiectului ei despre cărți. După ce a învățat cum să gestioneze o colecție de obiecte, ea descoperă acum nevoia de a copia și reasigna obiecte fără a produce erori de memorie. În același timp, observă că fiecare carte 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

- **carte.h** – declararea clasei `Carte`;
- **carte.cpp** – implementarea metodelor clasei;
- **main.cpp** – testarea funcționalităților.

Fișierul `carte.h`

```
#ifndef CARTE_H
#define CARTE_H

class Carte {
    char* titlu;
    char* autor;
    int pagini;                         // numar de pagini
    float rating;                       // nota acordata de student (0{10)

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

public:
    // Constructori
    Carte();
    Carte(const char* titlu, const char* autor, const int pagini, const
    float rating);
    Carte(const Carte& other);            // constructor de copiere
    Carte& operator=(const Carte& other); // operator de atribuire

    // Destructor
    ~Carte();

    // Getteri const
    char* getTitlu() const;
    char* getAutor() const;
    int getPagini() const;
    float getRating() const;

    const int getCodUnic() const;

    // Setteri
    void setTitlu(const char* titlu);
    void setAutor(const char* autor);
    void setPagini(const int& pagini);
    void setRating(const float& rating);

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

#endif CARTE_H
```

Fișierul `main.cpp` (model)

```cpp
#include "carte.h"

int main() {

    // TODO 1: Creati un obiect c1 folosind constructorul cu parametri
    // si il afisati.

    // TODO 2: Creati un obiect c2 folosind constructorul de copiere.
    // Ex: Carte c2(c1);

    // TODO 3: Testati operatorul de atribuire (=)
    // Ex: c3 = c1;

    // TODO 4: Cititi numarul de carti n si alocati un vector dinamic.
    // Ex: Carte* v_carti = new Carte[n]

    // TODO 5: Cititi datele fiecarei carti (autor, pagini, titlu, rating)
    // si completati vectorul folosind metoda set sau constructorul
    // cu parametri.

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

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

    // TODO 8: Eliberati memoria si verificati cu Valgrind.
    // g++ -g main.cpp carte.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 numarCarti; // comun tuturor obiectelor
```

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

```
int Carte::numarCarti = 0;
```

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

```
// Exemplu:
Carte::Carte(const Carte& c) : codUnic(++numarCarti) {

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

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

    this->pagini = c.pagini;
    this->rating = c.rating;
}

// 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 `Carte` cu atributele: `rating`, `pagini`, `titlu`, `autor`.
2. (2p) Implementați constructorul de copiere și operatorul de atribuire.
3. (2p) Adăugați un atribut `const int codUnic` și un membru static `numarCarti`.
4. (1p) Afișați codul unic al fiecărei cărți.
5. (1p) Afișați numărul total de cărți create.
6. (1p) Testați totul cu `valgrind`.

Compilare și verificare

```
g++ -g main.cpp carte.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 4:

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

3. După acceptare, veți primi propriul repository, ex. `laborator-4-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 `carte.h`, `carte.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.