

Laborator 4 – Particularitățile clasei

Tema: Studentul Gamer 🎮 – Partea a II-a

Autor: ing. Eduard Donea

Poveste

Alex a ajuns la partea a doua a proiectului său despre jocuri video. 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 joc 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 atributul `const`;
 - implementarea unui contor comun tuturor obiectelor (membru `static`);
 - verificarea corectitudinii alocării și eliberării memoriei cu `valgrind`.
-

Structura fișierelor

- `jocvideo.h` – declararea clasei `JocVideo`;
- `jocvideo.cpp` – implementarea metodelor clasei;
- `main.cpp` – testarea funcționalităților.

Fișierul jocvideo.h

```
#ifndef JOCVIDEO_H
#define JOCVIDEO_H

class JocVideo {
    char* titlu;
    char* gen;
    int durata;           // durata medie de joc (minute)
    float scor;           // scor dat de student (0{10)

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

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

    // Destructor
    ~JocVideo();

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

    const int getCodUnic() const;

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

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

#endif JOCVIDEO_H
```

Fișierul main.cpp (model)

```
#include "jocvideo.h"

int main() {

    // TODO 1: Creati un obiect j1 folosind constructorul cu parametri
    // si il afisati.

    // TODO 2: Creati un obiect j2 folosind constructorul de copiere.
    // Ex: JocVideo j2(j1);

    // TODO 3: Testati operatorul de atribuire (=)
    // Ex: j3 = j1;

    // TODO 4: Cititi numarul de jocuri n si alocati un vector dinamic.
    // Ex: JocVideo* v_jocuri = new JocVideo[n]

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

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

    // TODO 7: Afisati numarul total de jocuri create folosind metoda statica.

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

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

```
int JocVideo::numarJocuri = 0;
```

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

```
// Exemplu:
JocVideo::JocVideo(const JocVideo& j) : codUnic(++numarJocuri) {

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

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

    this->durata = j.durata;
    this->scor = j.scor;
}

// 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 `JocVideo` cu atributele: `titlu`, `gen`, `durata`, `scor`.
2. (2p) Implementați constructorul de copiere și operatorul de atribuire.
3. (2p) Adăugați un atribut `const int codUnic` și un membru static `numarJocuri`.
4. (1p) Afișați codul unic al fiecărui joc.
5. (1p) Afișați numărul total de jocuri create.
6. (1p) Testați totul cu `valgrind`.

Compilare și verificare

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

Bonus GitHub Classroom

Studentii 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/e5IrDPg0>

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

```
git clone
↪ git@github.com:Laborator-P00-2025-2026/laborator-4-NumePrenume.git
cd laborator-4-NumePrenume
```

5. După ce ați adăugat fișierele `jocvideo.h`, `jocvideo.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.