Laborator 10 – Vectori de obiecte neomogene

Tema: Media Library 🎞

Autor: ing. Eduard Donea

Poveste

În era digitală actuală, fiecare utilizator deține o colecție vastă de conținut media: melodii, videoclipuri, fotografii, podcasturi și multe altele. Pentru a organiza și accesa rapid aceste fișiere, este util un sistem unitar care să permită încărcarea, afișarea și gestionarea lor într-un mod polimorfic.

În laboratorul de astăzi veți construi un sistem simplificat de „Media Library”, în care fișierele media (audio, video, imagine) sunt tratate uniform prin intermediul unei clase abstracte și sunt gestionate într-un playlist ce funcționează cu un vector de pointeri către obiecte neomogene.

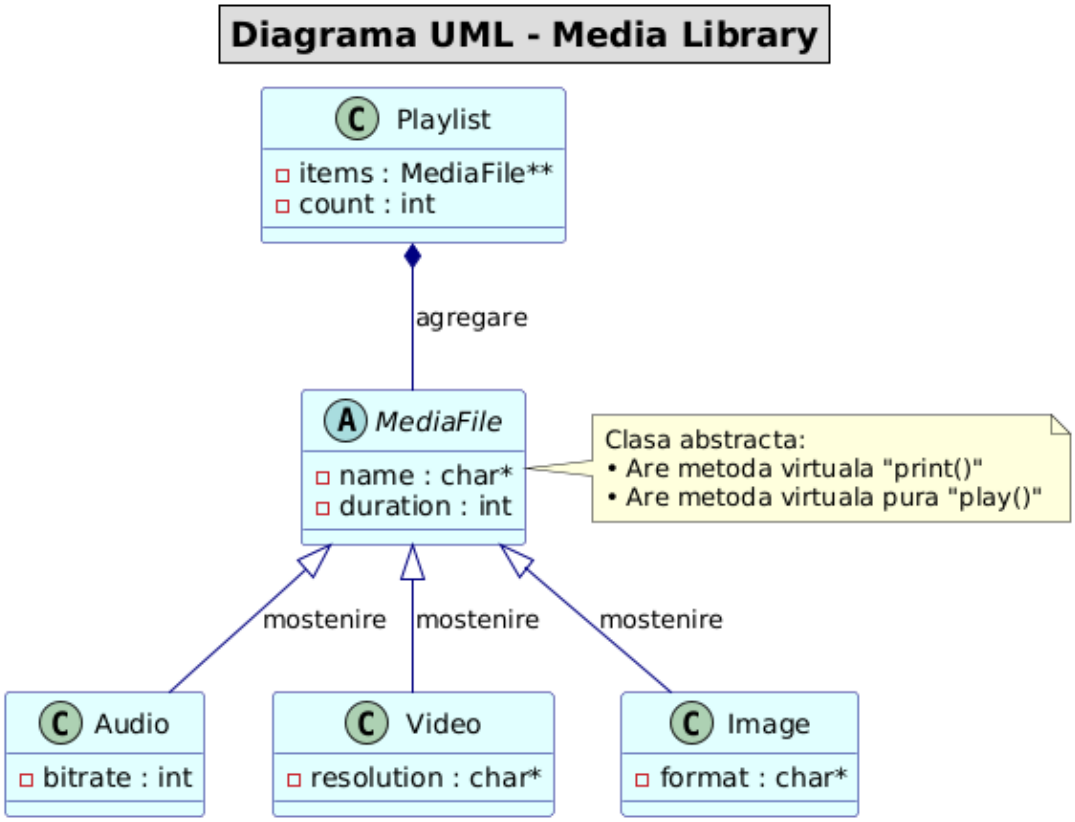
Scopul laboratorului

- înțelegerea folosirii pointerilor la clase abstracte;
- lucrul cu vectori de obiecte neomogene (polimorfism la runtime);
- diferențierea relațiilor `is-a` și `has-a`;
- folosirea unei metode virtuale pure într-un context real;
- suprascrierea metodei `print()` și integrarea cu `operator<<`.

Structura proiectului

- **MediaFile** – clasa abstractă (nume, durată, metode virtuale);
- **Audio**, **Video**, **Image** – clase derivate (tipul concret);
- **Playlist** – clasa care gestionează un vector de pointeri la `MediaFile`;
- **main.cpp** – programul principal unde sunt create fișierele și se rulează playlist-ul.

Figura 1 – Diagrama UML a sistemului Media Library



Clasa abstractă: MediaFile

Clasa `MediaFile` este superclasa tuturor fișierelor media din sistem. Ea definește atributele comune — numele fișierului și durata —, iar metoda `play()` este virtuală pură, obligând toate fișierele derivate să implementeze comportamentul de redare.
Metoda virtuală `print()` va fi folosită pentru suprascrierea operatorului `<<`, astfel încât orice fișier media să poată fi afișat în mod uniform.

Clase derivate: Audio, Video, Image

Audio reprezintă un fișier audio simplu. Metoda `play()` va afișa un mesaj specific redării de melodii.

Video reprezintă un fișier video și poate avea un comportament mai complex la redare (ex: rezoluție, subtitrări). Pentru laborator este suficient un mesaj reprezentativ.

Image reprezintă o fotografie. Deoarece imaginile nu se „redau", ci se „afișează", metoda `play()` va semnala deschiderea unei imagini.

OBSERVAȚIE: Puteți implementa orice alte tipuri media doriți (GIF, Podcast, Document etc.).

Clasa `Playlist`

Clasa `Playlist` gestionează un vector de pointeri la `MediaFile`.
Ea nu știe ce tip concret de media adaugă (Audio, Video, Image), ceea ce permite folosirea polimorfismului la runtime.
Metoda `playAll()` parcurge vectorul și apelează polimorfic `play()`, simulând o sesiune multimedia completă.

Fișierul `main.cpp`

```cpp
int main() {
    // TODO: creati cateva fisiere media (Audio, Video, Image)
    // TODO: creati un obiect Playlist
    // TODO: adaugati fisierele in vector
    // TODO: apelati playAll() si operator<<
    // TODO: eliberati memoria
    return 0; }
```

Cerințe (10p)

1. (3p) Implementați clasa abstractă `MediaFile` (constructori, destructor, metode virtuale).
2. (3p) Implementați clasele derivate `Audio`, `Video`, `Image` și comportamentul lor.
3. (2p) Implementați clasa `Playlist` și gestionați corect vectorul de pointeri.
4. (1p) Suprascrieți operatorul `<<` pentru afișarea fișierelor media.
5. (1p) Realizați o sesiune completă de redare în `main.cpp` folosind polimorfism.

Cerință Bonus (1p) – Gestionare avansată a unui Playlist multimedia

Pentru punctaj suplimentar, implementați în clasa `Playlist` următoarele funcționalități avansate de manipulare a colecțiilor de fișiere media:

```
void sortBy(const char* criteria);
// criterii valide: "name", "duration", "type"

MediaFile** filterBy(const char* type);

bool removeByName(const char* name);
```

Reguli obligatorii

- **Sortarea trebuie implementată manual**, folosind Bubble Sort, Selection Sort sau alt algoritm simplu (nu `std::sort`).
- **Filtrarea** trebuie să creeze și să returneze un **vector nou**, alocat dinamic, care conține doar elementele de tipul cerut. Playlist-ul original NU se modifică.
- **Metoda removeByName()** trebuie să:
  - caute elementul după nume;
  - elibereze memoria obiectului șters;
  - realoce vectorul intern și să îl compacteze corect;
  - returneze `true` dacă ștergerea a reușit, `false` altfel.

- Trebuie tratate cazurile speciale:
  - playlist gol;
  - tip inexistent sau nume inexistent;
  - playlist cu un singur element.
- Orice memorie alocată trebuie eliberată corespunzător (se verifică cu `valgrind`).

Compilare și rulare

```
g++ -g *.cpp -o lab10
valgrind ./lab10
```

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.

Pașii 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 10:

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

3. După acceptare, veți primi propriul repository, ex. `laborator-10-NumePrenume`.
4. Clonați repository-ul local:

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

   ---

5. După ce ați adăugat fișierele, 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.