Laborator POO : Mini-Steam

322ABb

12/11/2024

1 Introducere

Mircea, un student entuziast de la Facultatea de Automatica si Calculatoare, este un mare pasionat de jocuri video si tehnologie. De-a lungul anilor, a petrecut nenumarate ore explorand universurile fascinante oferite de platforme mari de jocuri, cum ar fi Steam, si si-a dorit mereu sa inteleaga cum functioneaza in spate o astfel de platforma. Inspirat de dorinta sa de a invata si de a construi ceva propriu, s-a hotarat sa creeze o platforma de jocuri mai mica, pe care a numit-o "mini-Steam".

Acum, Mircea are o idee clara despre structura generala a platformei si a inceput sa lucreze la un proiect de baza. Pentru a-si transforma ideea in realitate, a conceput un plan detaliat cu clasele si functionalitatile necesare: va avea o baza de date cu jocuri, va permite adaugarea si gestionarea acestora, precum si administrarea utilizatorilor care folosesc platforma. Totul este schitat si structurat, dar el are nevoie de ajutor pentru implementare.

Acesta este momentul in care intrati voi in scena. Sarcina voastra este sa-l ajutati pe Mircea sa construiasca platforma mini-Steam de la zero, aplicand cunostintele de Programare Orientata pe Obiecte. Veti lucra impreuna la structurarea claselor, la gestionarea mostenirii si agregarii datelor si, in final, la implementarea unui mic meniu care sa permita navigarea intre functionalitatile de baza ale platformei.

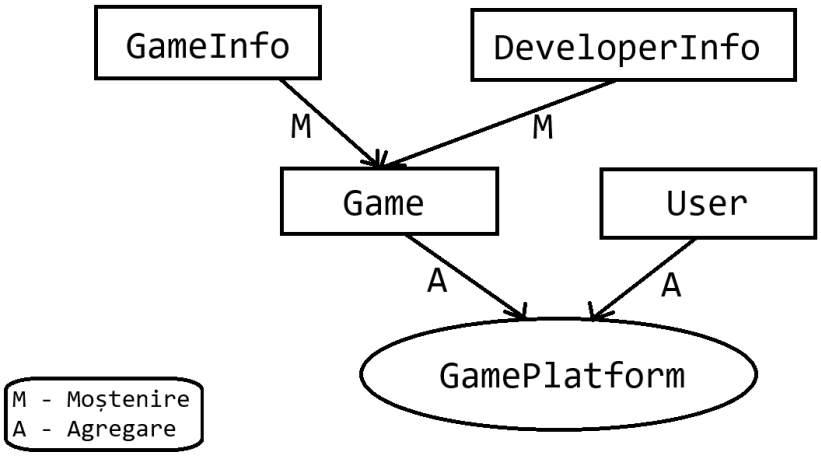
Prin acest proiect, veti exersa implementarea unor concepte fundamentale de programare orientata pe obiecte intr-un context captivant si practic. Sa incepem si sa-l ajutam pe Mircea sa-si transforme ideea intr-o platforma functionala!

2 Structura proiectului

Proiectul contine mai multe clase care reprezinta elementele esentiale ale unei platforme de jocuri:

- **GameInfo**: contine informatii despre joc (**title** si **genre**).
- **DeveloperInfo**: contine informatii despre dezvoltatorul jocului (**developerName**).
- **Game**: reprezinta un joc, mostenind din **GameInfo** si **DeveloperInfo**.
- **User**: reprezinta un utilizator al platformei (**username**).
- **GamePlatform**: clasa principala care gestioneaza o lista de **Game** si **User** prin agregare (**games**, **users**).
- **Utilities.hpp**: defineste functia `deepCopy` pentru a ajuta la copierea profunda a sirurilor de caractere.

3 Vizualizare Grafica a Ierarhiei Claselor



4 Fisierele .hpp

4.1 Utilities.hpp

Explicatie pentru functia deepCopy(): Functia `deepCopy` gestioneaza copierea profunda a unui sir de caractere. Folosim `char*& dest` pentru a modifica direct pointerul de destinatie, asigurandu-ne ca memoria este eliberata corespunzator si ca pointerul `dest` indica spre o zona de memorie nou alocata care contine continutul din `src`.
Ce face `inline`? Cand o functie este marcata ca `inline`, aceasta sugereaza compilatorului sa inlocuiasca apelurile functiei cu codul efectiv al functiei, pentru a evita costul apelului de functie. Asta poate imbunatati performanta pentru functiile mici si des utilizate. Cu toate acestea, compilatorul nu este obligat sa aplice aceasta optimizare chiar daca functia este marcata `inline` si va decide in functie de context daca este sau nu avantajos.

```
#ifndef UTILITIES_HPP
#define UTILITIES_HPP

#prgma warning (disable : 4996)

#include <cstring>

inline void deepCopy(char*& dest, const char* src) {

    delete[] dest; // Eliberare memorie anterioara

    if (src) {

        dest = new char[strlen(src) + 1];
        strcpy(dest, src);

    } else {

        dest = nullptr;

    }
}

#endif // UTILITIES_HPP
```

4.2 GameInfo.hpp

```cpp
#ifndef GAMEINFO_HPP
#define GAMEINFO_HPP

#include <iostream>
#include "Utilities.hpp"

class GameInfo {

protected:
    char* title;
    char* genre;

public:
    GameInfo(const char* title = "", const char* genre = "");
    GameInfo(const GameInfo& other);
    GameInfo& operator=(const GameInfo& other);
    ~GameInfo();

    void displayInfo() const;

    friend std::ostream& operator<<(std::ostream& out, const GameInfo& info);
    friend std::istream& operator>>(std::istream& in, GameInfo& info);

};

#endif // GAMEINFO_HPP
```

4.3 DeveloperInfo.hpp

```cpp
#ifndef DEVELOPERINFO_HPP
#define DEVELOPERINFO_HPP

#include <iostream>
#include "Utilities.hpp"

class DeveloperInfo {

protected:
    char* developerName;

public:
    DeveloperInfo(const char* developerName = "");
    DeveloperInfo(const DeveloperInfo& other);
    DeveloperInfo& operator=(const DeveloperInfo& other);
    ~DeveloperInfo();

    void displayDeveloper() const;

    friend std::ostream& operator<<(std::ostream& out, const DeveloperInfo& developer);
    friend std::istream& operator>>(std::istream& in, DeveloperInfo& developer);

};

#endif // DEVELOPERINFO_HPP
```

4.4 Game.hpp

```cpp
#ifndef GAME_HPP
#define GAME_HPP

#include "GameInfo.hpp"
#include "DeveloperInfo.hpp"

class Game : public GameInfo, public DeveloperInfo {

public:
    Game(const char* title = "", const char* genre = "", const char* developerName = "");

    void displayGameDetails() const;

    friend std::istream& operator>>(std::istream& in, Game& game);

};

#endif // GAME_HPP
```

4.5 User.hpp

```cpp
#ifndef USER_HPP
#define USER_HPP

#include <iostream>
#include "Utilities.hpp"

class User {

private:
    char* username;

public:
    User(const char* username = "");
    User(const User& other);
    User& operator=(const User& other);
    ~User();

    void displayUserInfo() const;

    friend std::ostream& operator<<(std::ostream& out, const User& user);
    friend std::istream& operator>>(std::istream& in, User& user);

};

#endif // USER_HPP
```

4.6 GamePlatform.hpp

```cpp
#ifndef GAMEPLATFORM_HPP
#define GAMEPLATFORM_HPP

#include <iostream>
#include <cstring>
#include "Game.hpp"
#include "User.hpp"

class GamePlatform {

private:
    char* platformName;

    Game** games;
    int gameCount;
    int gameCapacity;

    User** users;
    int userCount;
    int userCapacity;

public:
    GamePlatform(const char* platformName);
    ~GamePlatform();

    void addGame(const Game& game);
    void addUser(const User& user);

    void displayGames() const;
    void displayUsers() const;

};

#endif // GAMEPLATFORM_HPP
```

5 Fisierele .cpp

Vom apela functia deepCopy in implementare.

5.1 GameInfo.cpp

```cpp
#include "GameInfo.hpp"

GameInfo::GameInfo(const char* title, const char* genre) : title(nullptr), genre(nullptr) {
    //Copiere profunda cu deepCopy
    deepCopy(this->title, title);
    deepCopy(this->genre, genre);
}

GameInfo::GameInfo(const GameInfo& other) : title(nullptr), genre(nullptr) {
    //Copiere din other in this
    deepCopy(this->title, other.title);
    deepCopy(this->genre, other.genre);
}

GameInfo& GameInfo::operator=(const GameInfo& other) {
    if (this != &other) {
        //Copiere profunda
        deepCopy(this->title, other.title);
        deepCopy(this->genre, other.genre);
    }

    return *this;
}

GameInfo::~GameInfo() {
    //Eliberare memorie
    delete[] title;
    delete[] genre;
}

void GameInfo::displayInfo() const {
    std::cout << "Title: " << title << ", Genre: " << genre << std::endl;
}

std::ostream& operator<<(std::ostream& out, const GameInfo& info) {
    out << "Title: " << info.title << ", Genre: " << info.genre;

    return out;
}

std::istream& operator>>(std::istream& in, GameInfo& info) {
    char temp[100];

    std::cout << "Enter game title: ";
    in >> temp;
    deepCopy(info.title, temp);

    std::cout << "Enter genre: ";
    in >> temp;
    deepCopy(info.genre, temp);

    return in;
}
```

5.2 DeveloperInfo.cpp

Este asemanator cu GameInfo.cpp

5.3 Game.cpp

Imbina GameInfo si DeveloperInfo (HINT: Apelati metodele deja definite)

5.4 User.cpp

Este asemanator cu GameInfo.cpp si DeveloperInfo.cpp

5.5 GamePlatform.cpp

```cpp
#include "GamePlatform.h"

//TODO: Declarati constructorul cu valorile initiale urmatoare:
// -> Pentru gameCount si userCount 0
// -> Pentru gameCapacity si userCapacity >= 2
// Alocati corect memorie pentru cei doi vectori
GamePlatform::GamePlatform(const char* platformName)  {

}

//TODO: Implementati destructorul si asigurati-va ca ati sters TOATE obiectele
GamePlatform::~GamePlatform() {

}


//TODO: Implementati metoda addGame tinand cont de acesta aspecte:
// Verificati capacitatea sa nu fie mai mare de cat cantitatea:
// -> Cat timp verificarea e valida, cresteti capacitatea (ex: dublati-o)
// -> Creati si alocati memorie pentru vectorul de jocuri noi
// -> Copiati jocurile existente in vectorul nou alocat
// -> Eliberati memoria veche si actualizati pointer-ul
// Dupa verificare, adaugati noul joc si incrementati indexul
void GamePlatform::addGame(const Game& game) {

}

//TODO: Se procedeaza ca la addGame
void GamePlatform::addUser(const User& user) {

}

void GamePlatform::displayGames() const {

    std::cout << "Platform: " << platformName << "\nGames available:\n";

    for (int i = 0; i < gameCount; i++) {
        //TODO: Apelati metoda corespunzatoare (prin games)
        std::cout << "----------------------" << std::endl;
    }

}

void GamePlatform::displayUsers() const {

    std::cout << "\nUsers on platform:\n";

    for (int i = 0; i < userCount; i++) {
        //TODO: Apelati metoda corespunzatoare (prin users)
        std::cout << "----------------------" << std::endl;
    }

}
```

5.6 main.cpp

```cpp
#include "GamePlatform.h"
#include "Game.h"
#include "User.h"

int main() {

    //TODO: Creati un obiect de tip GamePlatform

    int choice;

    do {

        std::cout << "\n--- Mini-Steam Platform ---\n";
        std::cout << "1. Add a Game\n";
        std::cout << "2. Add a User\n";
        std::cout << "3. Display All Games\n";
        std::cout << "4. Display All Users\n";
        std::cout << "0. Exit\n";
        std::cout << "\nChoose an option: ";

        std::cin >> choice;

        if (choice == 1) {

            //TODO: Creati un obiect de tip Game (ex: Game newGame;)
            std::cout << "//Enter game details//\n\n";
            //TODO: Cititi de la consola (ex: std::cin >> newGame;)
            //si adaugati-l la platforma (ex: platform.addGame(newGame);)

        }
        else if (choice == 2) {

            //TODOL Creati un obiect de tip User
            std::cout << "//Enter user details//\n\n";
            //TODO: Citire de la consola si adaugare in platforma (similar ca la Game)

        }
        else if (choice == 3) {

            std::cout << "\n--- List of Games ---\n";
            //TODO: Apelati metoda corespunzatoare

        }
        else if (choice == 4) {

            std::cout << "\n--- List of Users ---\n";
            //TODO: Apelati metoda corespunzatoare

        }
        else if (choice != 0) {

            std::cout << "Invalid choice. Please try again.\n";

        }

    } while (choice != 0);

    std::cout << "\nExiting Mini-Steam...\n\n Goodbye!\n";

    return 0;

}
```

Punctaj:

- Implementare clase - 5p
- Implementare main - 3p
- Funcționalitate - 1p
- Oficiu - 1p